Software Safety Analysis of a Flight Guidance System

Alan C. Tribble, Steven P. Miller, and David L. Lempia

{actribbl, spmiller, dllempia}@rockwellcollins.com

Rockwell Collins, Inc., 400 Collins Rd, NE
Cedar Rapids, IA 52402 USA

Abstract

This document summarizes the safety analysis performed on a Flight Guidance System (FGS) requirements model. In particular, the safety properties desired of the FGS model are identified and the presence of the safety properties in the model is formally verified. Chapter 1 provides an introduction to the entire project, while Chapter 2 gives a brief overview of the problem domain, the nature of accidents, model based development, and the four-variable model. Chapter 3 outlines the approach, both for the traditional safety analysis techniques used in the early stages of the process and for the formal methods techniques used in the latter stages. Chapter 4 presents the results of the traditional safety analysis techniques, (Functional Hazard Assessment, Fault Tree Analysis, …), and illustrates how the hazardous conditions associated with the system trace into specific safety properties. Chapter 5 presents the results of the formal methods analysis technique – model checking – that was used to verify the presence of the safety properties in the requirements model. Finally, Chapter 6 summarizes the main conclusions of the study, first and foremost that model checking is a very effective verification technique to use on discrete models with reasonable state spaces. Additional supporting details are provided in the appendices.

Acknowledgements

This work was supported, in part, by the NASA Aviation Safety Program under contract NCC01-001 with the NASA Langley Research Center.

Table of Contents

List of Figures

List of Tables

1 Introduction

Air traffic is predicted to increase ten-fold by the year 2016. Along with the increase in traffic will be a proportionate increase in accidents, [1]. Unless we can reduce the accident rate from its current level, the increase in air traffic alone will account for a major hull loss every week to ten days, somewhere in the world.

Reducing the current aviation accident rate is a daunting task that will require a concerted effort on many fronts. It is virtually guaranteed that the development of new computer systems will play a key role in meeting this goal. Many of these systems will be more complex than any built to date. Just as mechanical engineering has invested in computer aided design tools to build today's skyscrapers and airframes, systems and software engineering will need to invest in new tools to meet the challenge of building the next generation of avionics systems. The goal of the "Methods and Tools for Flight Critical Systems" project, a cost-sharing effort jointly funded by the NASA Langley Research Center and Rockwell Collins, Inc. is to extend our software engineering infrastructure so that we will be able to safely deploy the avionics systems of the future with confidence.

The problem domain chosen for the study was a Flight Guidance System (FGS). The FGS is software centric function responsible for generating roll and pitch guidance values used by the Flight Control System (FCS). As such, it is an excellent candidate for an in depth study. In order to move the analysis as far upstream in the life cycle as possible our analysis used a Model Based Development (MBD) approach. In MBD, a model of the system requirements is one of the first products generated early in the life cycle of a system. The starting point for our safety analysis is therefore a requirements model for the FGS.

Most software engineering curriculums emphasize the techniques used to design and develop systems to meet certain functional requirements. Very little emphasis is placed on examining the possible consequences of failure. However, when dealing with safety-critical systems, such as those used in the aviation industry, having a firm understanding of these possible failure modes is essential. Traditional safety analysis techniques such as Functional Hazard Assessment (FHA), Fault Tree Analysis (FTA), and Failure Mode Effects Analysis (FMEA) have a long track record of being applied successfully to hardware intensive systems. It is therefore logical to extend these proven methods to software intensive systems. However, given the differences between hardware and software it is also appropriate to investigate new methods that may be required to analyze more complex, safety-critical systems. Consequently, this study has investigated the use of formal methods, model checking and theorem proving, in addition to the traditional safety analysis techniques.

One of the key benefits to this approach is that it moves some of the testing upstream from finished code to the actual requirements themselves. That is, we have used formal methods techniques to validate the requirements themselves and not just the code that is generated from the requirements. As such, we believe that the use of formal methods can increase confidence in the safety of the final product.

The chapters that follow outline our efforts to perform a comprehensive safety analysis on the requirements model of a FGS. Chapter 2 provides background information, including a description of the problem domain, the nature of accidents, model based development, and the four-variable model. Chapter 3 outlines the approach used, both for the traditional safety analysis techniques and the formal methods techniques. Chapter 4 summarizes the results of the traditional safety analysis, which is a listing of the properties of the FGS requirements model that relate to safety. Chapter 5 summarizes the results of the formal methods analysis, and illustrates how model checking has been used to verify the presence of all of the safety properties in the requirements model. Finally, Chapter 6 summarizes the main conclusions and identifies possible future directions.

2 Background

This chapter provides a brief overview of the problem domain, the nature of accidents, model based development, and the four-variable model paradigm.

2.1 The Problem Domain

One of the objectives of this project is to perform an extensive analysis of a safety critical system that reflects the complexity of an actual product. The aviation domain provides a number of excellent candidates and the avionics system of a typical regional jet aircraft was chosen because of its safety critical nature and its inherent complexity. As shown in Figure 1, the avionics architecture is comprised of many individual subsystems. Featured in this diagram are the Flight Control System (FCS) and Flight Management System (FMS). The FCS in turn is composed of a Flight Guidance System (FGS), Flight Director (FD), Auto-Pilot (AP), and Auto-Throttle (AT).

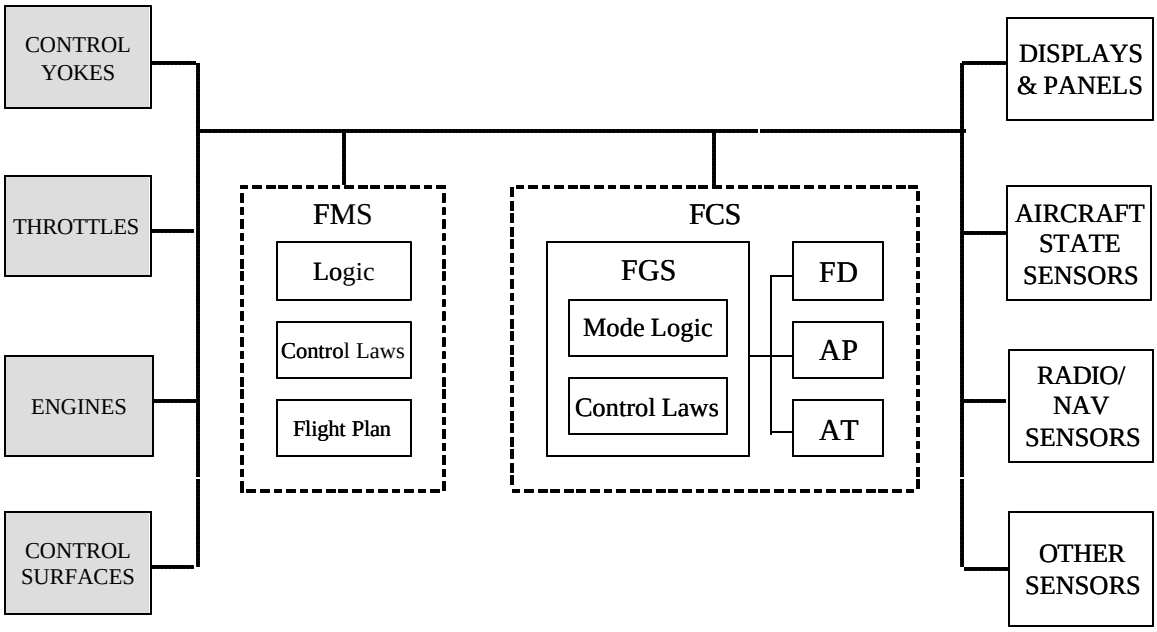

Figure 1. The High Level Architecture of an Avionics System.

The FGS is decomposed into discrete and continuous elements called the mode logic and the flight control laws. The flight control laws are continuous functions that compare the measured state of the aircraft (position, speed, attitude, altitude, …) to the desired state and generate guidance commands to minimize the difference between the two. Most systems generate only roll and pitch attitude commands, but some also generate throttle (speed) commands. The mode logic is a set of discrete algorithms that select the appropriate flight control laws for use any time the system is active. The FGS mode logic was targeted as the problem of interest for this project due to its discrete nature, and the fact that it is mainly implemented in software.

The FGS guidance values are passed to the FD, AP, and AT. The FD converts the FGS guidance values into visual cues that are shown to the flight crew via the displays in the cockpit. In most systems, the FD is a notional entity whose functionality is contained within the FGS and the Displays & Panels subsystem. The AP converts the FGS guidance values into commands used to control the movement of the actual flight control surfaces. Similarly, the AT converts the FGS guidance values into commands used to control the setting of the aircraft thrusters. Both the AP and AT are separate, standalone elements with dedicated hardware.

2.1.1 FGS Functional Requirements

The FGS is a software function assigned responsibility for four main areas as shown in Table 1. The first function, compute flight guidance steering commands, is performed by the flight control laws. The second function, select and indicate flight guidance mode, is the responsibility of the mode logic. The third and fourth functions, control the FD and AP, are a combination of discrete logic and hardware functionality. A fifth function, control the AT, is considered outside the scope of this project. Each of the functions identified in Table 1 is elaborated on in the following paragraphs. In a later section, the functional requirements will be the starting point for the safety analysis that will follow.

Table 1. Typical FGS Functions.

Ref.	Function
1	Compute Flight Guidance Steering Commands
1.1	Compute Roll and Pitch Guidance Values
2	Select and Indicate Flight Guidance Mode
2.1	Select Flight Guidance Mode
2.2	Indicate Flight Guidance Mode
3	Control FD
3.1	Control Display of FD Guidance Cues
4	Control AP
4.1	Control Transfer of Flight Guidance Values to AP
4.2	Indicate Transfer of Flight Guidance Values to AP
4.3	Control AP Engagement / Disengagement
4.4	Indicate AP Engagement / Disengagement

Compute Flight Guidance Steering Commands

The FGS must correctly compute the roll and pitch steering commands for all flight modes. That is, the continuous flight control laws must generate the correct numerical values that will be used by the FD, AP, and AT. The flight control laws themselves will not be considered in scope for this analysis. That is, it will be assumed that the flight control laws always generate the correct numerical values.

Select and Indicate Flight Guidance Mode

The FGS must correctly select the active and armed modes of operation and must correctly identify these modes to the flight crew. This is entirely the responsibility of the software and is the focus of the majority of the safety analysis for this project. The modes are selected by Boolean logic based on inputs received from other systems. As will be seen, the modes are identified to the flight crew via outputs to the Displays and Panels subsystem. The primary effort of the safety analysis will be to ensure that the correct mode is selected based on the inputs, and that the correct mode is indicated based on the outputs.

Control FD

It must be possible to turn the FD on when off, and vice versa. Because the FD is a notional entity the responsibility for activating / de-activating it is assigned to software. The safety analysis must also address the possibility of incorrectly activating, or de-activating, the FD.

Control AP

Like the FD, it must be possible to turn the AP on when off, and vice versa. Unlike the FD, much of the responsibility for engaging / disengaging the AP is assigned to hardware and not software. Nevertheless, the software does have some responsibility for verifying that conditions are valid for engaging the AP and for initiating AP disengagement in certain circumstances. The safety analysis must therefore address the possibility of incorrectly allowing AP engagement, or of incorrectly commanding AP disengagement. Similarly, the AP must be aware of which side is the pilot flying side so it knows which FGS is the master. This will also be addressed in the safety analysis.

2.1.2 FGS Modes

FGS modes are usually segregated into two categories: lateral and vertical. The lateral modes of operation control the horizontal motion of the aircraft by adjusting the aircraft roll (the angle of rotation about the axis from the aircraft's nose to its tail.) The vertical modes of operation control the vertical motion of the aircraft by adjusting the aircraft pitch (the angle of rotation about the axis parallel to the aircraft's wings.) The third axis of rotation is called yaw (the angle of rotation about the axis perpendicular to both roll and pitch.) Yaw is used mainly to adjust the orientation of the aircraft to ensure smooth flight. Although yaw damping is a part of many AP's it is independent of the mode of operation and is not addressed in this analysis. Typical lateral and vertical modes of operation for a regional jet aircraft are shown in Table 2 and Table 3, respectively. Note that if a lateral mode is active a vertical mode must also be active, and vice versa.

Table 2. Lateral Modes.

Mode	Description
Roll (ROLL)	The default mode of operation where the aircraft will hold a fixed roll angle. This is the default lateral mode and is always active when the FGS is on and no other lateral mode is active.
Approach (APPR)	Used for precision approaches when the aircraft is attempting to capture, or has captured, the specified navigation source - either LOC, VOR, or NAV (FMS). This mode is selected manually by pressing the APPR button on the FCP.
Go Around (GA)	The aircraft will hold a pre-set reference heading. Manually selected by the flight crew by pressing the GA button on the control yokes.
Heading (HDG)	The aircraft will track the heading displayed on the PFD. This mode is selected manually when the HDG button on the FCP is pressed.
Navigation (NAV)	Used for en route and non-precision approaches. The aircraft will acquire and track the navigation source displayed on the PFD. This mode is selected manually when the NAV button on the FCP is pressed.

Table 3. Vertical Modes.

Mode	Description
Pitch (PTCH)	The default mode of operation where the aircraft will maintain a fixed pitch angle. This is the default vertical mode and is always active when the FGS is on and no other vertical mode is active.
Altitude Hold (ALT)	The aircraft will maintain the pressure altitude. This mode is activated manually when the ALT button on the FCP is pressed.
Altitude Select (ALTS)	The aircraft will capture the PreSelect Altitude set by the preselect dial on the FCP, and will then track that altitude using the Barometric Altitude. Altitude select mode is normally armed when other modes are active. However, it is cleared by the selection of Altitude Hold, Approach, or Go Around.
Approach (APPR)	Used for precision approaches. The aircraft will acquire and track the Glide Slope (GS). This mode is selected manually by pressing the APPR button on the FCP.
Flight Level Change (FLC)	The aircraft will acquire and track an Indicated Air Speed (IAS) or Mach reference speed by adjusting the aircraft pitch and will ascend or descend as throttles are increased or decreased.
Go Around (GA)	The aircraft will hold a pre-set reference pitch angle. Manually selected by flight crew by pressing the GA button on the control yokes.
Vertical Speed (VS)	The aircraft will maintain the specified vertical speed (climb or descent) reference, defined by the vertical speed dial on the FCP.

2.1.3 FGS Interfaces

An intermediate level of the FGS is shown in Figure 2. On an actual aircraft the FGS will interface to several other subsystems, such as the Attitude/Heading Reference System (AHRS), Air Data System (ADS), Flight Control Panel (FCP), Flight Management System (FMS), the Navigation Radios, Primary Flight Displays (PFD), and - indirectly - the Display Control Panel (DCP). Normally, only one FGS (the pilot flying side) is active, with the other FGS operating as a silent, hot spare. In this situation, the active FGS provides guidance values to the AP and each FD, and provides mode annunciations to each PFD. However, in some modes, such as Approach and Go Around, both units are active and generate guidance values for their own FD and mode annunciations for their own PFD independently. In these independent modes of operation, both sets of guidance values are provided to the AP, which first verifies that they agree within a pre-defined tolerance value. If in agreement, the values are averaged and executed. If in . If in

Figure

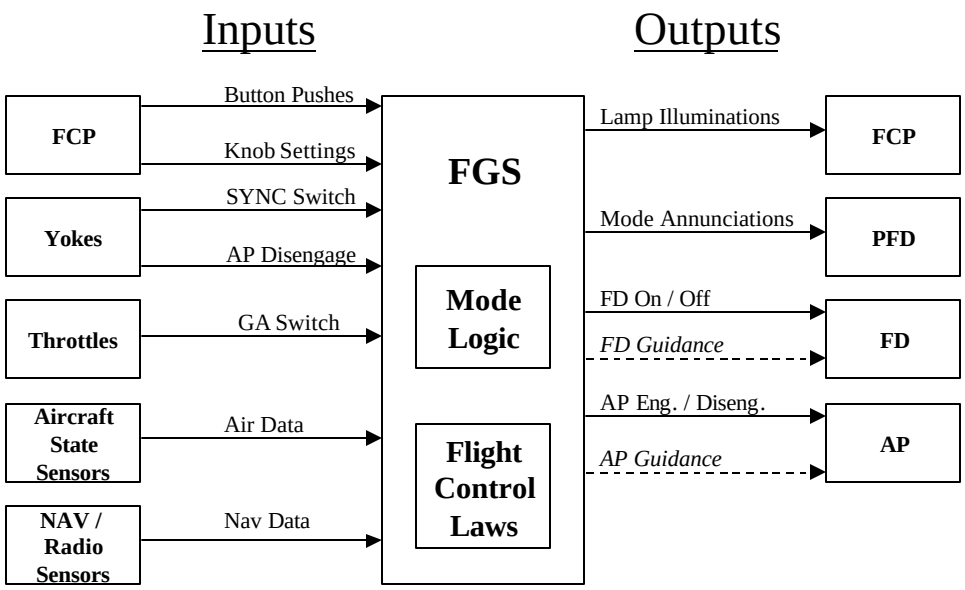

Figure 3. Overview of a Flight Guidance System and its Interfaces.

Flight Control Panel (FCP)

The FCP is used to select modes of operation for the FGS, to turn the FD on or off, to engage or disengage the Autopilot AP, and to input certain reference values. An example FCP is shown in Figure 4.

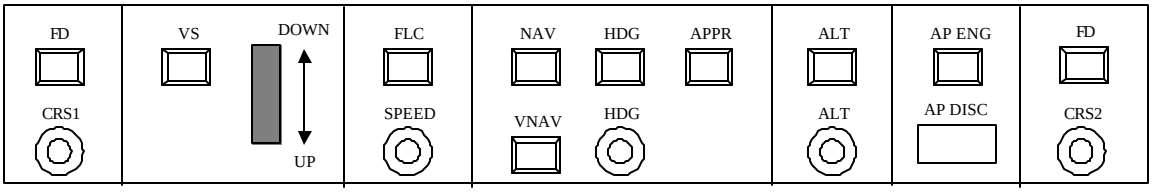

Figure 4. The Flight Control Panel (FCP) is Used to Select FGS Modes.

From left to right the FCP switches are:

- The pilot's button for turning the FD on and off and a knob for adjusting the reference heading.

- A button for selecting the vertical speed (VS) mode and a wheel for increasing / decreasing the value of vertical speed desired. Note that the wheel can also be used for increasing / decreasing the pitch of the aircraft when in PTCH mode.

- A button for selecting the Flight Level Change (FLC) mode and a knob for selecting the desired airspeed.

- Buttons for selecting lateral navigation (NAV), heading hold (HDG), approach (APPR), and vertical navigation (VNAV) modes and a knob for selecting the heading desired.

- A button for selecting altitude (ALT) mode and a knob for selecting the desired (preselect) altitude.

- A button to engage the AP and a toggle switch to disengage the AP.

- The co-pilot's button for turning the FD on and off and a knob for adjusting the heading.

In some cases, the FCP also supplies feedback to the crew, indicating selected modes by lighting lamps located on either side of a selected mode's button. These are not shown in Figure 4, but will be addressed in the analysis.

Primary Flight Display (PFD)

As its name implies, the PFD is the most important device for relaying information to the flight crew. As shown in Figure 5, the PFD displays information on the FGS modes of operation, FD guidance cues, and AP engagement.

Annunciations

The FGS lateral and vertical mode annunciations are displayed at the top of the PFD. In this example, "ROLL" is the active lateral mode and "PTCH" is the active vertical model. The armed modes, the ones that will be activated in the near future once certain criteria are met, are annunciated below the active modes. In this example, "ALTS" is the armed vertical mode and there is no armed lateral mode.

Flight Director (FD)

The FD is the colored wedge "^" shown above the aircraft position "^". The position of the FD guidance cues above or below the aircraft position indicates the pitch correction recommended by the FGS, while the angle of the guidance cues indicates the roll correction recommended. In this example, the FD is recommending a 7.5 degree wings level climb. The aircraft itself is flying level, 0 degrees pitch and 0 degrees roll.

Pilot Flying Indicator

The transfer switch indicates which side is responsible for flying the aircraft. When in a dependent mode, (all modes except Approach and Go Around), the FGS on the Pilot Flying (PF) side is the master and the FGS on the Pilot Not Flying (PNF) side is the slave.

Autopilot (AP)

When the AP is on the letters "AP" appear directly above the pilot flying indicator as shown. Note that when the AP is engaged the AP flies the aircraft to the roll and pitch values displayed

by the FD and the aircraft position "^" would fit snugly into the FD "^", (in Figure 5 the AP has just been engaged and has not yet had time to fly the aircraft into its desired orientation.)

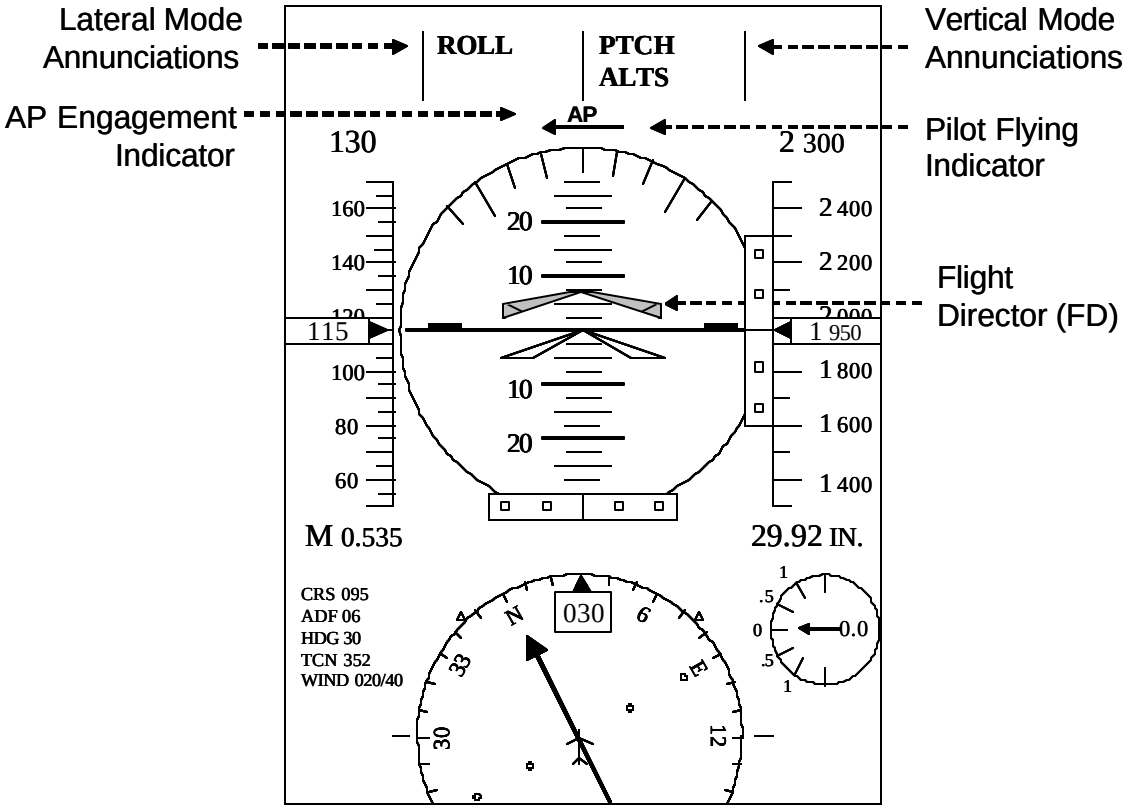

Figure 5. An Example Primary Flight Display (PFD), Used to Display Active Modes and Reference Values in Addition to Basic Flight Data.

2.2 The Nature of Accidents

Although thorough knowledge of the nature of accidents is not necessary in order to appreciate the value of our results, a high level understanding is helpful in order to see how this same approach could be applied in a larger context. Underlying our analysis is an assumption about the nature of accidents as shown in Figure 6. The definitions used in this accident model are in general agreement with IEEE standards, [2].

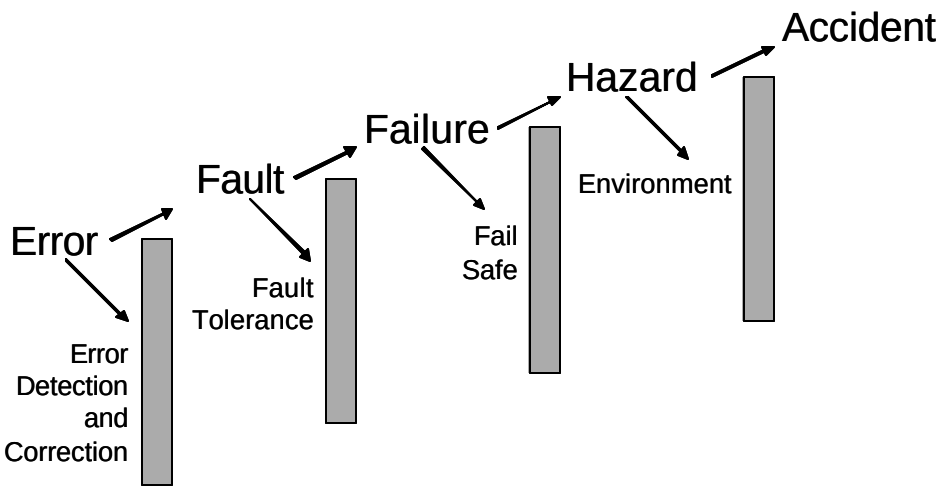

Figure 6. The Accident Model.

As shown, errors – mistakes in requirements, design, development, or test – can be the root cause of an accident. These errors may arise anywhere in the project life cycle. Ideally, all errors will be detected and corrected before the system is placed in operation. In reality, some errors will escape detection (and the corrective measures used to fix some others may actually introduce new errors) and make it into the final product as defects. It is important to note that even if designed, manufactured, and tested to excruciating standards a perfectly functioning system may still contain logic errors if the requirements used to develop the system are incorrect. Regardless of the source, some of these errors may propagate to the next stage and be manifested during operation as faults. Other faults may arise due to transient events, such as single event upset, or physical failures of correctly designed components. Fault tolerance design techniques may be used to contain the faults, but some faults may propagate to the next stage and result in system level failures – a loss of functionality. At this stage fail-safe design techniques may again halt the process, but some failures may not be contained and will place the system in a hazardous condition – a state that has the potential to result in an accident. The final factor that determines whether or not an accident occurs is the condition of the surrounding environment, (local terrain, other aircraft, weather, …). A hazardous condition coupled with "good" environmental conditions may not result in an accident, and is only an incident. If "bad" conditions are present the result will be an accident. Thus, an error may be manifested as a fault, a fault may result in a failure, a failure may place the system in a hazardous condition, and a hazardous condition may result in an accident.

Our safety analysis therefore focuses on defining the hazards, failures, faults, and errors that could lead to accidents. As shown in later sections, our analysis will use a combination of standard techniques, (e.g., Fault Tree Analysis and Failure Mode Effects Analysis), in combination with non-traditional, yet very powerful, formal methods techniques.

It is difficult to separate safety requirements from functional requirements. Similarly, it is difficult to separate software from hardware or data issues. Much of the complexity of the real system is related to the interaction between safety and functional requirements, and between software, hardware, and data. As a result, a comprehensive safety analysis must address these system level interactions. That is, an analysis of the mode logic of a single FGS must examine the properties of the mode logic in the context of the overall system.

Subsystem analysis must examine both the hardware and software aspects of the design, as well as possible interactions between the hardware and software. For the system examined here numerous failures may occur. For example, lights may burn out on the FCP. The result would be an inconsistent annunciation to the flight crew as the PFD would indicate one mode of operation, but the corresponding mode lights on the FCP would fail to illuminate. However, the same inconsistent annunciation may result from the FGS mode logic failing to command illumination of the lamp. There is no way for the flight crew to tell the difference between these two failures simply by looking at the displays. More complex failures, involving strings of hardware-software failures, are also possible. For example, if the mode logic commands an incorrect FCP lamp illumination AND the FCP lamp is burned out no indication to the flight crew will be made and the system may appear to be working perfectly when two simultaneous failures have occurred. It is important to emphasize that the analysis performed here focuses on software failures. Consequently, the results obtained in the remainder of this study should dovetail nicely with traditional hardware-centric safety analysis performed on similar systems and should pave the way for validating the formal methods techniques utilized in the later sections.

2.3 Model Based Development (MBD)

The process followed to develop the FGS model is often referred to as Model Based Development (MBD). MBD can best be understood by contrasting it with a traditional development process. The traditional life cycle process for product development follows a "V" shape as shown in Figure 7. First, requirements for the system are solicited, usually in the form of imprecise English statements. Based on the understanding of the requirements a design, or architecture, capable of meeting the requirements is created. Following the design, code is then developed that – it is hoped – is capable of delivering the functionality desired. Subsections of code are then integrated together and extensively tested, at the unit, subsystem, and system level, to verify that the required functionality is indeed present and that no unexpected behavior is manifested. Finally, the system is placed in operation and maintained throughout its operational life.

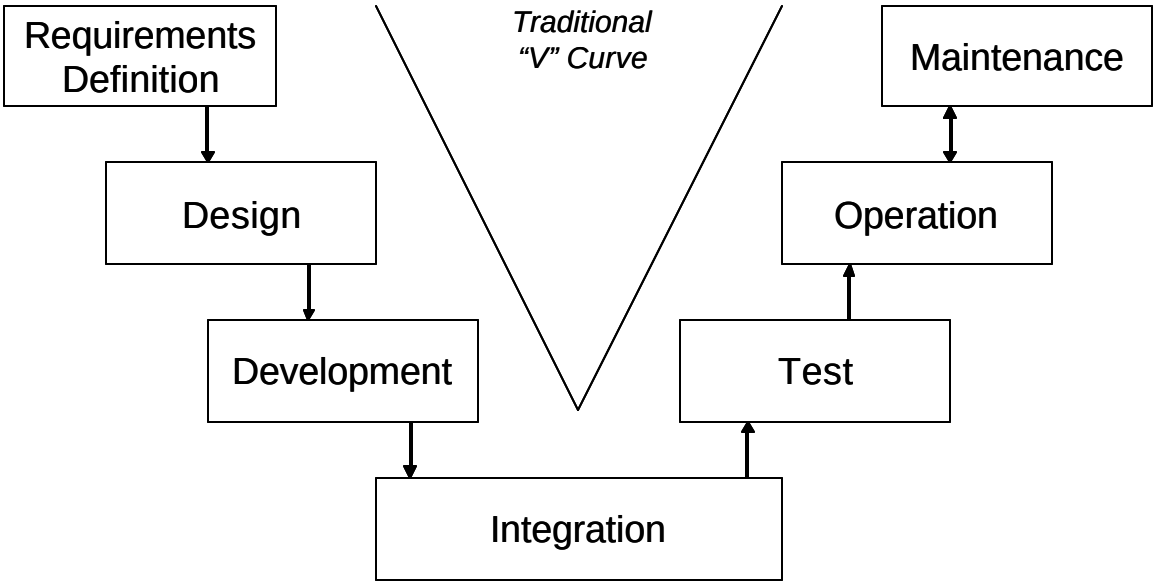

Figure 7. The Traditional Life Cycle Development Process.

While this approach is capable of developing very complex, safety-critical systems, it leaves much to be desired. Much of the time, and cost, of development is incurred in the middle stages of the process. As a result, most of the errors that are discovered are found after the design is chosen and code has been developed when it is difficult, and time consuming, to correct them without introducing new errors.

MBD attempts to address these concerns by shifting much of the validation work to the requirements definition and design stages while at the same time automating the transition from design to operation. As the name implies, the first step in MBD is the construction of a model which accurately reflects the required behavior of the system. Preferably, this is a requirements (black box functional) model that is independent of the system architecture. This omits implementation bias and makes the model more widely applicable. Once the functional behavior has been modeled it can be analyzed for the correct behavior and then used as the launch point for future stages. In particular, the model can be automatically translated into code, minimizing the possibility that errors are introduced and eliminating the time required for hand coding. The model can also be used to automatically generate the required test cases, again minimizing the possibility that errors are introduced and eliminating the time required for manual generation. This shifts the life cycle curve from the traditional "V" shape to a streamlined "Y" shape by replacing the labor intensive design – integration – test portions with automated generation of code and test cases, Figure 8.

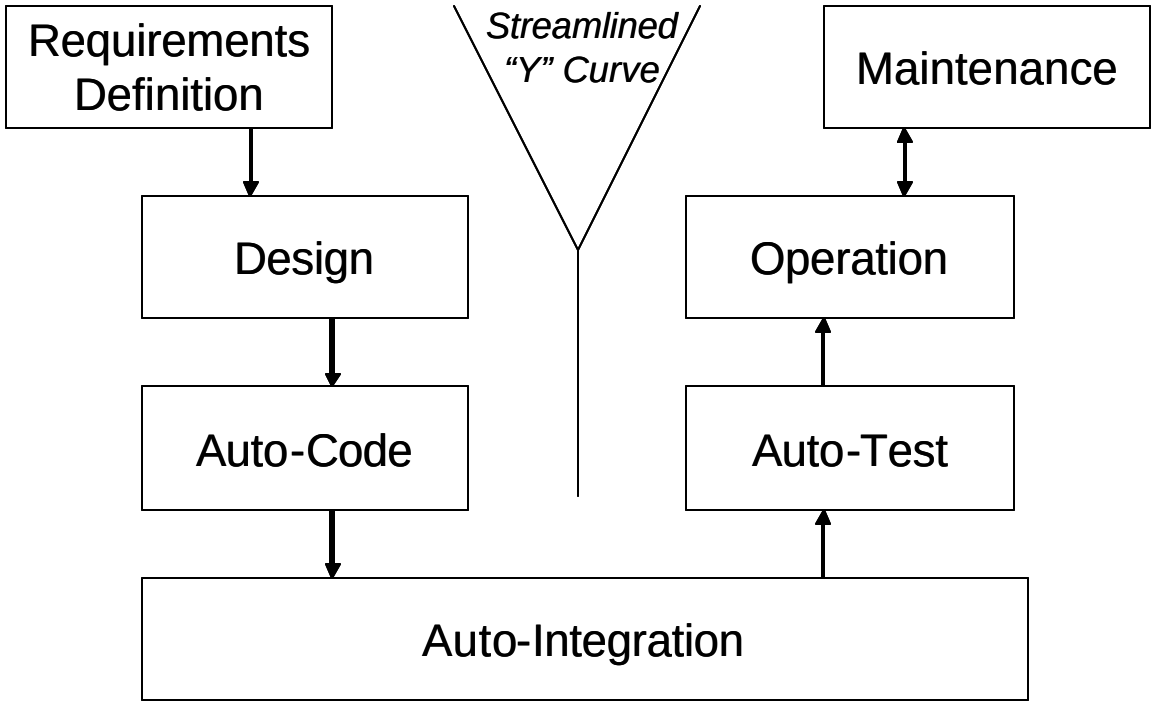

Figure 8. The Model Based Development Life Cycle Process.

We have followed a MBD paradigm for this project by creating a requirements model for the FGS in the Requirements State Machine Language without Events ($RSML^{-e}$). Details on the $RSML^{-e}$ notation are provided in Section 2.5. One unique aspect of this project is that the safety analysis will be performed not on the software, or hardware, but on the requirements themselves.

2.4 The Four-Variable Model

The FGS model was structured after the original four-variable model proposed by Parnas and Madey, [3]. As shown in Figure 9, the variables in this model are continuous functions of time and consist of:

- *Monitored variables* (MON) in the environment that the system observes and responds to;
- *Controlled variables* (CON) in the environment that the system is to control;
- *Input variables* (INPUT) through which the system senses the monitored variables; and
- *Output variables* (OUTPUT) through which the system changes the controlled variables.

For example, monitored values might be the actual altitude of an aircraft and its airspeed while controlled variables might be the position of a control surface such as an aileron or the displayed value of the altitude on the primary flight display. The corresponding input and output values would be the ARINC-429 bus words that the software reads, or writes, to sense these quantities.

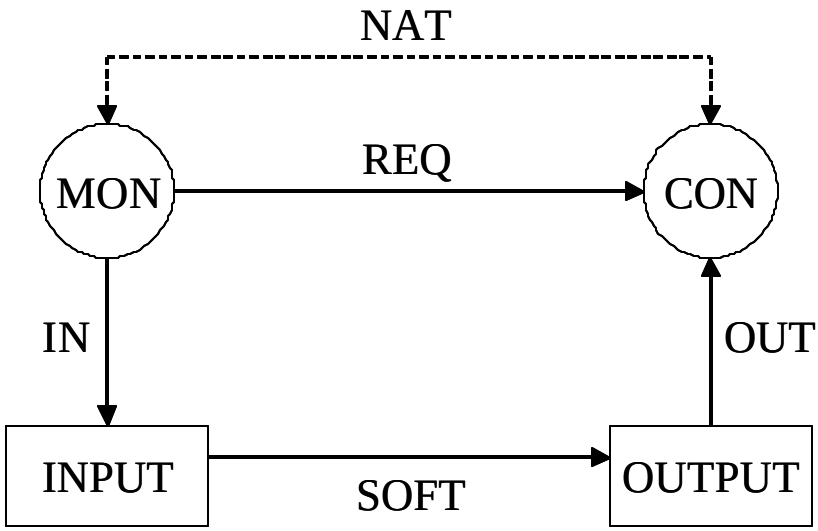

Figure 9. The Four-Variable Model

To complete the specification, four mathematical relations are defined between the variables:

- **NAT** defines the natural constraints imposed by the environment, such as the maximum rate of climb of an aircraft;
- **REQ** defines the system requirements, specifying how the controlled variables are to respond to changes in the monitored variables;
- **IN** defines the relationship of the monitored variables to the input variables; and
- **OUT** defines relationship of the output variables to the controlled variables.

NAT and **REQ** describe how the controlled variables should change in response to changes in the monitored variables and define the subsystem view of the specification. **NAT** describes how the environment (the monitored and controlled variables) behaves in the absence of the system to be built, while **REQ** describes how the environment (the controlled variables) is to be constrained by the system. These relationships can be specified with mathematical precision, making them ideal for specifying safety-critical systems. The hardware interfaces surrounding the software are modeled by the **IN** and **OUT** relations that define how the input and output variables the software interacts with are related to the monitored and controlled environmental variables. Specification of the **NAT**, **REQ**, **IN**, and **OUT** relations implicitly bounds the allowed behavior of the software, shown in Figure 9 as **SOFT**, without specifying its design.

One of the great advantages of this model is that it explicitly defines the system boundary through the identification of the monitored and controlled variables. If MON and CON are chosen correctly, **IN** and **OUT** will change only as the underlying hardware changes. At the same time, **REQ** changes only in response to changes in the system requirements. Since customer driven changes and hardware driven changes often arise for different reasons, this helps to make the system more robust in the face of change.

Much additional work has been done to support the use of the four-variable model in real applications. For example, the Naval Research Lab has worked to formalize the four-variable

model in SCR and to provide supporting tools, [4]. Also, Rockwell Collins has described an extension to the four-variabi 4.5 0 TD TD 0 Tc 0 6atrting tos g

the controlled variables, CON'. This shift in perspective is possible only because of the tacit assumption that **REQ** and **REQ'** are identical. In verifying the properties of the FGS mode logic, we also verified that the MON' variables were correctly constructed from the input variables and that the output variables were correctly constructed from the CON' variables.

2.5 Formal Methods Tools

The term *Formal Methods* refers to a variety of mathematical modeling techniques that are applicable to computer system (software and hardware) design. In much the same way that aeronautical engineers may make use of computational fluid dynamics (CFD) to predict how a particular airframe design will behave in flight, computer scientists use formal methods to predict the behavior of software or hardware. Two common formal methods tools are model checkers and theorem provers, [6 - 11].

Model checking is a technique that examines each and every possible state of the system to verify that a desired property, (e.g., safety, liveness, functional, …), holds in the model. The state space search is guaranteed to terminate if the model is finite and if a counterexample is found, it is known that the property does not hold. Theorem proving is a technique in which properties of the model are derived using the rules of mathematical logic. That is, proving a theorem is the process of verifying the existence of a property from the specification by repeatedly applying transformations known to preserve correctness until the theorem is derived.

2.5.1 The RSML^{-e} Specification Language

The starting point for formal analysis is a formal specification language. RSML (Requirements State Machine Language) is a state-based specification language for modeling the behavior of process control systems that was developed by Nancy Leveson's group at the University of California at Irvine, [12]. One of the main design goals of RSML was readability and understandability by non-computer professionals such as end-users, engineers in the application domain, managers, and representatives from regulatory agencies. RSML was used to specify TCAS-II and this specification was ultimately adopted by the FAA as the official specification for TCAS-II. RSML was heavily influenced by Statecharts, [13], and uses a similar notion of explicit event propagation. In the course of developing the TCAS-II specification and the subsequent independent verification and validation effort, it became clear that the most common source of errors was this dependence on explicit events, [14]. To eliminate this problem, the Critical Systems group at the University of Minnesota developed RSML^{-e} (RSML without Events), [15]. As its name implies, RSML^{-e} eliminates the use explicit events and is a synchronous language. Other examples of synchronized languages include SCR, LUSTRE, (the kernel language for SCADE), and Esterel, [16 – 21]. RSML^{-e} is similar to another derivative of RSML, SpecTRM-RL, developed by the Safeware Engineering Corporation, but has a slightly different syntax and semantics and differs in the underlying modeling philosophy. RSML^{-e} runs in the "Nimbus" environment also developed at the University of Minnesota. This environment provides a framework simulating, visualizing, and analyzing RSML^{-e} specifications.

2.5.2 The NuSMV Model Checking System

Model checking is a formal verification technique that allows one to check for properties of a model through exhaustive exploration of the state space. This makes verification of properties highly automated and straightforward. However, state space explosion limits the size of the models that can be analyzed. Explicit state model checkers, such as SPIN, explicitly represent the states visited in a form that grows in proportion to the number of states being modeled. Symbolic model checkers represent the state space symbolically, often using Binary Decision Diagrams (BDDs). Both approaches are ultimately limited by the size of the state space, but symbolic model checkers can usually explore larger state spaces than explicit state model checkers, [6].

NuSMV is a symbolic model checker developed as a joint project between the Formal Methods group in the Automated Reasoning System Division at the Instituto Trintino di Cultura (ITC) - Center for Scientific and Technological Research (IRST), the Mechanized Reasoning Groups at the University of Genova and the University of Trento in Italy, and the Model Checking group at Carnegie Mellon University in the United States. NuSMV is a re-implementation and extension of SMV, [7], the first model checker based on BDDs. NuSMV has been designed to be an open architecture for model checking, which can be reliably used for the verification of industrial designs, as a core for custom verification tools, as a testbed for formal verification techniques, and applied to other research areas, [8].

2.5.3 The PVS Theorem Proving System

In contrast to model checkers, theorem provers apply rules of inference to a specification in order to derive new properties of interest. Theorem provers are generally harder to use than model checkers, requiring considerable technical expertise and understanding of the specification. However, theorem provers are not limited by the size of the state space. Also, some properties that cannot be easily specified using model checkers, such as comparing properties of two arbitrary states that are not temporally related, can be easily specified in the languages of most theorem provers.

PVS is an environment for specification and verification that has been developed at SRI International's Computer Science Laboratory. In comparison to other widely used verification systems such as HOL and the Boyer-Moore Prove, the distinguishing characteristic of PVS is that it supports a highly expressive specification language with a highly effective interactive theorem prover in which most of the lower-level proof steps are automated. The system consists of a specification language, a parser, a type checker, and an interactive proof checker. The PVS specification language is based on higher-order logic with a richly expressive type system so that a number of semantic errors in specification can be caught during type checking. The PVS prover consists of a powerful collection of inference steps that can be used to reduce a proof goal to simpler subgoals that can be discharged automatically by the primitive proof steps of the prover. The primitive proof steps involve, among other things, the use of arithmetic and equality decision procedure, automatic rewriting, and BDD-based Boolean simplification, [9 - 11].

3 Approach

This chapter provides an overview of the processes used to conduct the safety analysis on the FGS requirements model. The results from the traditional safety analysis techniques are presented in Chapter 4, while the results from the formal methods analysis techniques are presented in Chapter 5.

3.1 Traditional Safety Analysis Techniques

As shown in Figure 6, a necessary precursor to an accident is a hazardous condition. Traditional safety analysis therefore begins by defining the hazards associated with a system, determines their severity, and then attempts to identify the factors that can initiate the hazards.

3.1.1 Functional Hazard Assessment (FHA)

By definition, hazards are conditions that the system should avoid. Once the hazards are known it becomes possible to trace backwards from the hazards into the events that can cause them. Hazards are derived from functional failures, consequently the initial safety analysis efforts concentrate on defining the functionality required and analyzing the consequences of failure. This is accomplished in the Functional Hazard Assessment (FHA). The FHA is an informal process that is used to document hazards and determine their severity. The output of an FHA is a tabular listing of the hazards and their level of severity.

Safety is a system level problem. As a result, aviation safety standards ARP 4754 and ARP 4761 specify that safety analysis be performed both at the aircraft, or system, level and at the subsystem level, [22, 23]. The aircraft level hazards are generally very few, such as:

- Loss of Control (LOC)

When the LOC hazard is examined at the aircraft level, it will be found that a number of subsystem failures could give rise to it, such as hydraulic lines failures, control yoke failures, flight control surface failures, and so on. For the purposes of this study we will be concerned only with the failures of the FGS that can lead to aircraft level hazards.

It is important to note that not every hazard will be the result of hardware, or software, failures. Some can simply be the result of misuse. For example, two possible hazards associated with the AP are: i) Pilot initiated AP engagement into an uncertified condition; and ii) Failure to disengage the AP when entering an uncertified condition. To prevent this type of hazard, the AP hardware is designed such that it will not engage in an uncertified condition and will "cutout" if forced into an uncertified condition.

As stated before, we are interested in defining the hazards for the FGS itself. These hazards will derive from the functional requirements previously defined for the FGS. Based on the terminology defined by DO-178B and MIL STD 882, the FGS is categorized as a Level C

(Major) system, Table 4, [24, 25]. Although FGS failures that result in incorrect guidance may appear to be of a higher level of criticality, in reality a complete Flight Control System has enough checks and balances built in that the resulting effect on the flight crew is at most "significant" and not "extreme". It is routinely verified in flight tests that even torque-limited hardovers due to mechanical failure do not cause "hazardous" conditions. In comparison, the worst that a FGS mode logic failure could do would be to select the wrong flight control law, which in turn would cause deviation from the intended flight plan. The flight crew would sense this departure as part of their normal monitoring routine and would be able to correct for it with minimal effort.

Table 4. Hazard (Failure) Criticality Levels as Applied to Aircraft Design.

DO-178B	**Level E**	**Level D**	**Level C**	**Level B**	**Level A**
MIL STD 882	**NA**	**Category IV**	**Category III**	**Category II**	**Category I**
Classification Of Failure	**None**	**Minor**	**Major**	**Hazardous**	**Catastrophic**
Effect on Aircraft	No effect on operational capabilities or safety margin	Slight reduction in operational capabilities or safety margin	Significant reduction in operational capabilities or safety margin	Large reduction in operational capabilities or safety margin	Safe flight and landing prevented, usually with loss of aircraft
Effect on Passengers	Inconvenience	Physical discomfort	Physical distress, possibly including injuries	Serious or fatal injury to a small number of occupants	Multiple fatalities
Effect on Flight Crew	None	Slight increase in workload or use of emergency procedures	Physical discomfort or a significant increase in workload	Physical distress or excessive workload impairing ability to perform tasks	Fatalities or incapacitation
Interpreted Qualitative Probability	*NA*	*Probable*	*Remote*	*Extremely Remote*	*Extremely Improbable*
Interpreted Quantitative Probability	*NA*	*10^{-3} per flight hour*	*10^{-5} per flight hour*	*10^{-7} per flight hour*	*10^{-9} per flight hour*

3.1.2 Fault Tree Analysis (FTA)

Once the hazards have been identified it is necessary to trace backwards through the accident model to the failures / faults / errors that could initiate them. Fault Tree Analysis (FTA) is a top-down analysis technique that is used to identify the contributing elements (errors / faults / failures) that could precipitate the system level hazards identified, [26]. FTA is a feed-back technique in that one starts with the system level hazards and attempts to work backward by identifying all possible causes of the hazards. Although the name implies that the technique is limited only to "faults", it should be emphasized that FTA is a general, visual technique that is used to trace higher level events (such as hazards) down to their contributing events. These contributing events could be failures, or errors, in addition to faults.

The FTA is presented as a visual, tree-like structure where the various factors that contribute to a high level event are linked together. Typical FTA symbology is defined in Figure 11. As shown, the highest level event (hazard) is traced backward through various contributing events until the base event – the most fundamental thing that can go wrong – is identified. In an actual aircraft program, the FTA would start with the system level hazard, for example, Loss of Control, and include all aircraft systems (including the flight crew) that could potentially contribute to such a hazard. The objective for our project will be to identify all of the base events (errors) that could precipitate FGS functional failures, and hence hazardous conditions. These base events will form the general categories of the specific safety properties required of the FGS.

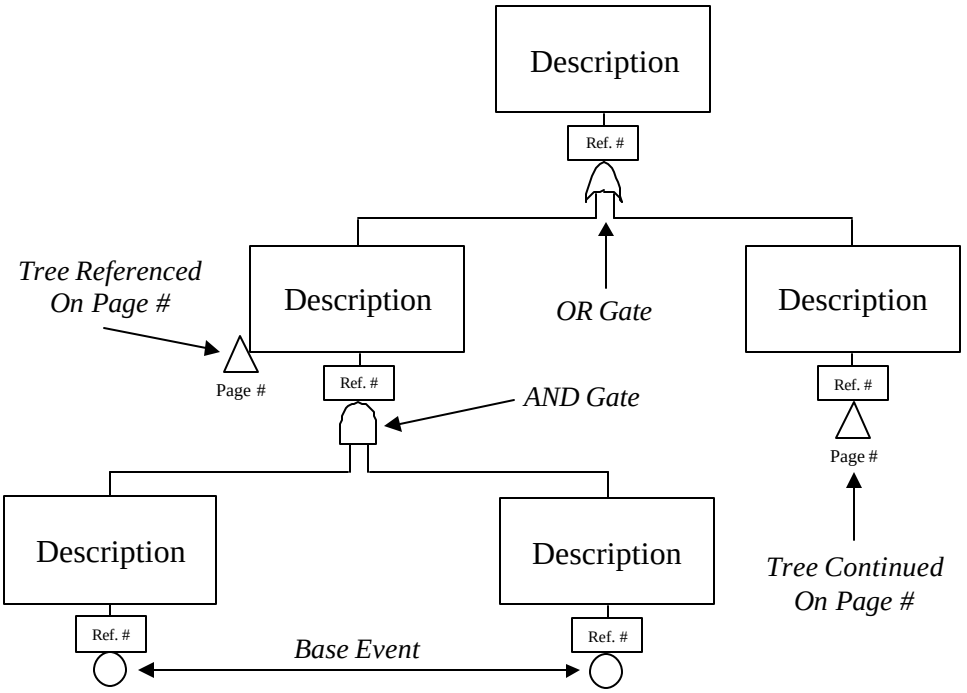

Figure 11. An Example of the Symbology Used in a Fault Tree Analysis (FTA).

3.1.3 Failure Mode Effects Analysis (FMEA)

Because the FTA is a top-down analysis technique it does not (directly) consider the possibility that errors could generate more than one hazard. For this reason a bi-directional approach, that utilizes both top-down and bottoms-up analysis, is often preferred. Failure Mode Effects Analysis (FMEA) is an bottom-up analysis procedure which documents all probable failures of a system, determines the effect of each failure on system operation, and ranks each failure according to severity, [26]. FMEA is a feed-forward technique in that the starting points for the analysis are possible failures, (or faults or errors), which are then traced forward to see if they have any impact on system safety (i.e., if they lead to potential hazards). The output of a FMEA is a tabular presentation that lists: a) failure mode; b) effects; and c) analysis. As with the FTA, the term FMEA should not imply that the results are limited to "failures". FMEA is a general analysis method that flows errors (or faults or failures) forward to hazardous conditions.

3.1.4 Safety Properties

From the FHA, FTA, and FMEA we will obtain a listing of base events that can contribute to hazardous conditions. This list of events is in essence a listing of the general categories of safety properties required of the FGS requirements model. The final step in the traditional safety analysis process is to articulate the specific manifestations of the general safety categories based on the capability of the FGS model. That is, rather than attempt to prove general properties such as "no errors in mode selection logic" the properties must be made very specific to the architecture of the system, such as "ALT mode is always activated when the ALT button is pressed". The next challenge is then to find some way of verifying that the requirements model does indeed manifest each of the safety properties. For this, we use formal methods.

3.2 Formal Methods Analysis

The use of formal methods in assessing safety involves four steps as shown in Figure 12. First, the system itself must be specified, or modeled, in a formal language. Second, the safety properties must also be defined formally. Third, since specifications are written in a notation designed for humans rather than formal analysis tools, both the specification and the safety properties must usually be translated into the notation of the formal methods tools. Finally, the analysis itself is conducted. A top-level overview of each of these steps is provided in the sections that follow.

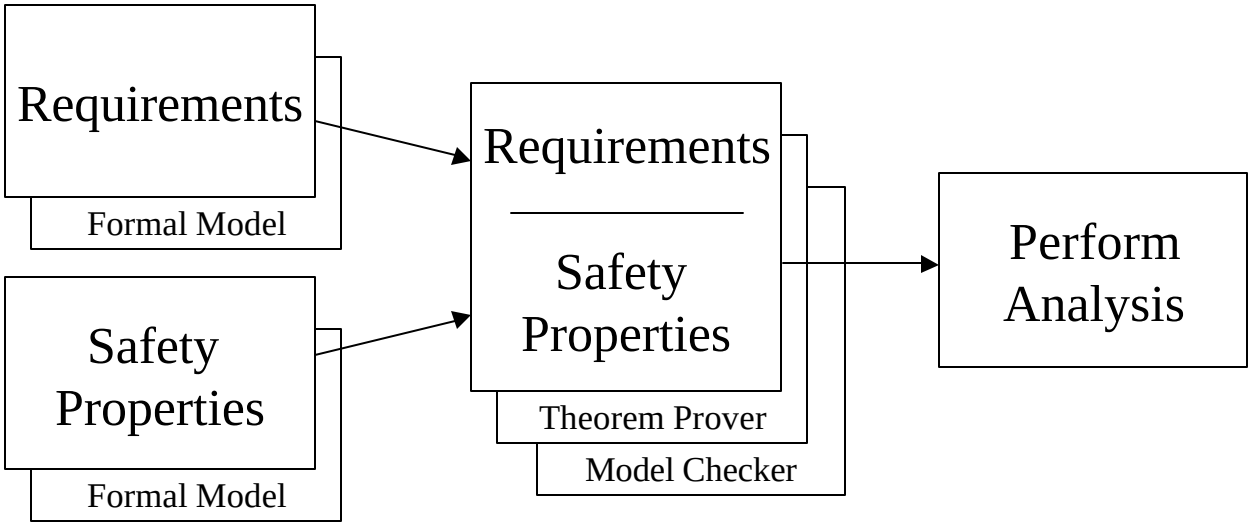

Figure 12. The Formal Methods Approach to Safety Analysis.

3.2.1 Modeling the Requirements

Since the first step in the use of formal methods is the development of a model there is a natural correlation between MBD and the use of formal methods. As was previously mentioned, the FGS requirements model has been generated in a formal language, RSML^{-e}. The FGS model is described in detail in a separate report and will not be discussed further here, [27].

3.2.2 Defining the Safety Properties

The next step in the safety analysis process is to formally define those properties of the software associated with safety. The safety properties themselves are identified by the traditional safety analysis techniques previously discussed. It should be noted that in most cases the "safety" requirements are simply the "functional" requirements that trace directly to hazards. As a result, our approach has focused on verifying requirements in general, without regard to their criticality.

3.2.3 Translating the Requirements Model and Safety Properties into Analysis Tools

The third step in the safety analysis process is to ensure that the requirements model and safety properties are both expressed in the same formal language. It is certainly possible to define the requirements and safety properties in the same language initially, but given the current state of the art this will rarely be the case. The requirements model will probably be developed in tools that are popular with developers such as RSML^{-e}, Esterel, SCADE, or Matlab. Safety properties will probably be defined in English prose. Even if the properties were defined in the requirements modeling tool, the modeling tool itself would probably lack the analysis capability and would require translation to a theorem prover or model checker to perform the actual analysis.

As part of this project, the University of Minnesota automated the translation of the model from RSML^{-e} to the notations of the NuSMV model checker, [16 - 18], and the PVS theorem prover, [19 - 21]. Building the translation capability is possible since both the origin and destination languages have a well defined semantics. The details on this translation are discussed separately in a companion report, [28]. The translation of the safety properties from English prose into SMV and PVS was done manually.

3.2.4 Conducting the Analysis

The process of verifying that the model possesses the desired properties proceeds differently depending on the tool selected. As will be discussed in Chapter 5, the use of a model checker is highly automated. Once the model and properties are defined the tool automatically performs a state space search to verify that each property is met. If a false counter-example is found it is then up to the user to determine why the failure occurred, and to then modify the requirements model, or property, accordingly. Additional work, such as ordering the state variables in the binary decision diagram, may be necessary for the verification to complete in a reasonable time.

As is discussed in a companion report, [29], the use of a theorem prover is not as highly automated and may require much more interactive guidance from their user. Nevertheless, once a proof is found, it can easily be re-verified if the model is updated and it can also be used as a starting point for more complicated proofs. The main advantage of theorem provers is that they are not sensitive to the size of the model and may often succeed where a model checker would fail due to the explosion of the state space. A wider variety of properties can also be specified with a theorem prover than with a model checker.

4 Traditional Safety Analysis Results

This chapter presents the results from the traditional safety analysis techniques used to generate the safety properties for the FGS requirements model.

4.1 Functional Hazard Assessment (FHA)

The FHA for the FGS requirements model is based on the FGS functional requirements shown in Table 1. The FHA for the failure to "compute flight guidance steering commands" is shown in Table 5. Note that the two functional failures are very similar, yet distinct. In general, failing to perform a function (loss of a capability) is less severe than incorrectly performing the function. Loss of guidance values would be minor in that the AP, or FD, should sense the absence of data and signal a disconnect of those systems. Incorrect guidance values is a major hazard in that it may cause the aircraft to drift from its intended flight plan producing a "significant reduction in safety margin". This could also require a "significant increase in workload" on the part of the flight crew to return the aircraft to its intended flight plan.

Table 5. Functional Hazard Assessment for Requirement: Compute Flight Guidance Steering Commands.

Ref.	Functional Failure (Hazard)	Critical Operational Phase	Aircraft Manifestation	Criticality	Comment
1.1.1	Loss of Guidance Values	Approach	Presence of No Computed Data (NCD) should signal FD and AP disconnect.	Minor	Becomes major hazard, equivalent to incorrect guidance, if disconnect fails.
1.1.2	Incorrect Guidance Values	Approach	Gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.	Major	No difference to the AP between loss of guidance and incorrect guidance.

The FHA for the failure to "select and indicate flight guidance mode" is shown in Table 6. The first two hazards, failure to select mode or incorrect mode selection, is minor in that the guidance values would either not be present, (loss of guidance), or correct for the active mode. The third hazard, loss of mode indication, is minor in that the flight crew would immediately notice the loss and disconnect the system. The second hazard, incorrect mode indication, is major because it could result in the aircraft deviating from its intended flight plan.

Table 6. Functional Hazard Assessment for Requirement: Select and Indicate Flight Guidance Mode.

Ref.	Functional Failure (Hazard)	Critical Operational Phase	Aircraft Manifestation	Criticality	Comment
2.1.1	Failure to Select Mode	All	If no mode is selected no data is generated, signaling disconnect.	Minor	Becomes Major Hazard if Disconnect Fails.
2.1.2	Incorrect Mode Selection	All	Flight guidance mode other than the one desired by the flight crew is armed or activated.	Minor	Assumes Guidance Values and Indications are Correct.
2.2.1	Loss of Mode Indication	Approach	Flight crew unable to determine mode and state of flight guidance resulting in manual disconnect and manual flying.	Minor	Loss of Mode Indication Less Serious Because Flight Crew Can Tell There is No Indication.
2.2.2	Incorrect Mode Indication	Approach	Gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.	Major	Assumes Guidance Values are Correct.

The FHA for the failure to "control the FD" is shown in Table 7. All of these hazards are minor in that they would be at most minor nuisances to the flight crew.

Table 7. Functional Hazard Assessment for Requirement: Control FD - Control Display of Flight Guidance Cues.

Ref.	Functional Failure (Hazard)	Critical Operational Phase	Aircraft Manifestation	Criticality	Comment
3.1.1	Unable to Activate FD	Approach	No FD guidance available. Manual flying.	Minor	-
3.1.2	Inadvertent FD Activation	All	FD guidance cues displayed without request.	Minor	-
3.1.3	Unable to Deactivate FD	All	FD guidance cues always displayed.	Minor	-
3.1.4	Inadvertent FD De-Activation	Approach	Absence of FD guidance cues noticed during check of primary flight data, manual flying.	Minor	-

The FHA for the failure to "control the AP" is shown in Table 8 and Table 9. As seen in Table 8, failure to "control and indicate transfer of flight guidance commands to AP" could result in the major hazard "incorrect indication of flight guidance transfer state". Similarly, failure to "control and indicate AP engagement" could result in the major hazard "incorrect AP engagement indication", Table 9.

It should be noted that we have implicitly assumed the FGS will be used in a Category II system, (one where precision approach and landing operations are conducted with a decision height of less than 200 feet, but more than 100 feet, and a runway visual range of less than 1200 feet). If the FGS were used in a Category III system, with smaller permissible values for the decision height and runway visual range, several failures could rise to "Hazardous" or "Level B" criticality. For a Category II system they are at most "Major".

Table 8. Functional Hazard Assessment for Requirement:
Control AP - Control and Indicate Transfer of Flight Guidance Commands to AP.

Ref.	Functional Failure (Hazard)	Critical Operational Phase	Aircraft Manifestation	Criticality	Comment
4.1.1	Loss of Transfer Control of Flight Guidance Data to AP	All	Flight crew unable to change "Pilot Flying" side FGS. Manual disconnect and manual flying.	Minor	-
4.1.2	Inadvertent Transfer of Flight Guidance Data to AP	Approach	Possible gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.	Minor	-
4.2.1	Loss of Flight Guidance Transfer State	All	Flight crew unable to determine "Pilot Flying" side. Manual disconnect and manual flying.	Minor	-
4.2.2	Incorrect Indication of Flight Guidance Transfer State	All	Incorrect "Pilot Flying" side indicated. Possible gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.	Major	Departure from references occurs only if pilot flying and pilot not flying have selected different navigation sources.

Table 9. Functional Hazard Assessment for Requirement: Control AP - Control AP Engagement.

Ref.	Functional Failure (Hazard)	Critical Operational Phase	Aircraft Manifestation	Criticality	Comment
4.3.1	Unable to Engage AP	All	AP unavailable. Manual flying.	Minor	-
4.3.2	Unable to Disengage AP	All	Flight crew detects AP engagement by either AP annunciation on PFD or resistance to control column / wheel inputs. Manual hardware disconnect of AP and manual flying.	Minor	The AP system provides independent disengagement mechanisms.
4.3.3	Inadvertent AP Engage-ment	Approach	Flight crew detects AP engagement by either AP annunciation on PFD or resistance to control column / wheel inputs. Manual hardware disconnect of AP and manual flying.	Minor	-
4.3.4	Inadvertent AP Disengage-ment	Approach	Disconnect should sound aural and visual alarms, alerting flight crew of the need for manual flying.	Minor	If no warning is provided, the criticality becomes Major.
4.4.1	Loss of AP Engage-ment Indication	All	If engaged, engagement noticed by resistance to control column / wheel inputs. If disengaged, departure from references noticed during check of primary flight data. Result is manual disconnect and manual flying.	Major	Engagement is indicated both on the PFD and FCP. Failure to send indication upon activation would be immediately recognized and is a Minor hazard.
4.4.2	Incorrect AP Engage-ment Indication	Approach	If engaged, engagement noticed by resistance to control column / wheel inputs. If disengaged, departure from references noticed during check of primary flight data. Result is manual disconnect and manual flying.	Major	-

The FHA identified five major hazards and fifteen minor hazards as shown in Table 10. The FHA has therefore confirmed that the FGS is a Level C (Major) system. Upon further analysis it can be seen that the major hazards *Loss of AP Engagement Indication* and *Incorrect AP Engagement Indication* are functionally equivalent. *Loss of AP Engagement Indication* is really a subset of *Incorrect AP Engagement Indication*, the case where the AP is engaged and annunciated as disengaged, so it may be dropped from further consideration. The remaining four major hazards will be examined further in the next sections.

Table 10. Summary of Hazards Identified in the Functional Hazard Assessment.

Functional Requirement	Ref	Hazard	Criticality
Compute Roll and Pitch Guidance Values	1.1.1	Loss of Guidance	Minor
	1.1.2	*Incorrect Guidance*	*Major*
Select Flight Guidance Mode	2.1.1	Failure to Select Mode	Minor
	2.1.2	Incorrect Mode Selection	Minor
Indicate Flight Guidance Mode	2.2.1	Loss of Mode Indication	Minor
	2.2.2	*Incorrect Mode Indication*	*Major*
Control Display of FD Guidance Cues	3.1.1	Unable to Activate FD	Minor
	3.1.2	Inadvertent FD Activation	Minor
	3.1.3	Unable to De-Activate FD	Minor
	3.1.4	Inadvertent FD De-Activation	Minor
Control Transfer of Flight Guidance Values to AP	4.1.1	Loss of Transfer Control of Guidance Data to AP	Minor
	4.1.2	Inadvertent Transfer of Guidance Data to AP	Minor
Indicate Transfer of Flight Guidance Values to AP	4.2.1	Loss of Transfer State Indication	Minor
	4.2.2	*Incorrect Transfer State Indication*	*Major*
Control AP Engagement / Disengagement	4.3.1	Unable to Engage AP	Minor
	4.3.2	Unable to Disengage AP	Minor
	4.3.3	Inadvertent AP Engagement	Minor
	4.3.4	Inadvertent AP Disengagement	Minor
Indicate AP Engagement / Disengagement	4.4.1	*Loss of AP Engagement Indication*[*]	*Major*
	4.4.2	*Incorrect AP Engagement Indication*	*Major*

[*] *Loss of AP Engagement Indication* and *Incorrect AP Engagement Indication* are functionally equivalent.

4.2 Fault Tree Analysis (FTA)

The FTA uses as its starting point each of the major hazards identified by the FHA. On an actual system all of the hazards would be included, but on this proof of concept study only the major hazards will be addressed. Because our ultimate objective is to map each hazard to "specific" safety properties associated with the FGS it is necessary to have a greater level of insight into the construction of FGS model itself. The specification for the FGS model is organized into the eight functional categories shown in Table 11. Such functional decomposition is a natural byproduct of systems engineering and breaks the full "system" level requirements into manageable "subsystem" sized piece. The FTA will therefore map each hazard into finer and finer levels of contributing events until one of the functional categories identified in Table 11, or its equivalent in a non-FGS element, has been reached.

Table 11. The FGS Model Functional Categories.

Category	Description
Annunciation	Monitors and controls the PFD mode annunciations and FCP lamp illuminations.
FD Selection	Monitors and controls the FD selection state.
Pilot Flying Transfer	Monitors and controls the PF and PNF status of the FGS.
Independent / Active	Monitors and controls the independent / dependent (master-slave), and active / inactive (standby) status of the FGS.
AP Engagement	Monitors and controls the AP engagement state.
Mode Selection	Monitors and controls the lateral and vertical mode selection.
Synchronization	Synchronizes the inactive FGS to active FGS.

The top levels of the FTA for the hazard *Incorrect Guidance* are shown in Figure 13. The fault tree first splits into "Incorrect AP Guidance" and "Incorrect FD Guidance" because the AP and the FD both receive guidance values from the FGS, but are implemented independently of one another. At the next level, the "Internal Error" event acknowledges the fact that the AP, or FD, themselves may corrupt correct data values provided to them by the FGS. The event "Incorrect Guidance Values Received From FGS" addresses the possibility that the FGS may pass incorrect values. These incorrect guidance values may in turn be due to "Communications Channel" or "Output Overwhelms" hardware failures, in addition to the FGS internal event, "FGS Sends Incorrect Guidance Values".

The lower levels of the FTA are shown in Figure 14. Recall that in the actual system two identical FGS units are in operation at any time, Section 2.1.3. In most cases, one FGS is active and the second operates as an inactive, hot spare. For some modes, Approach and Go Around, each FGS is considered active. As a result, the primary concern is that the active FGS could provide incorrect guidance values. However, if the inactive FGS outputs guidance data it could overwrite the guidance data from the active FGS. Therefore, the FTA must consider the possibility that incorrect FGS guidance values could originate from either the active or inactive FGS.

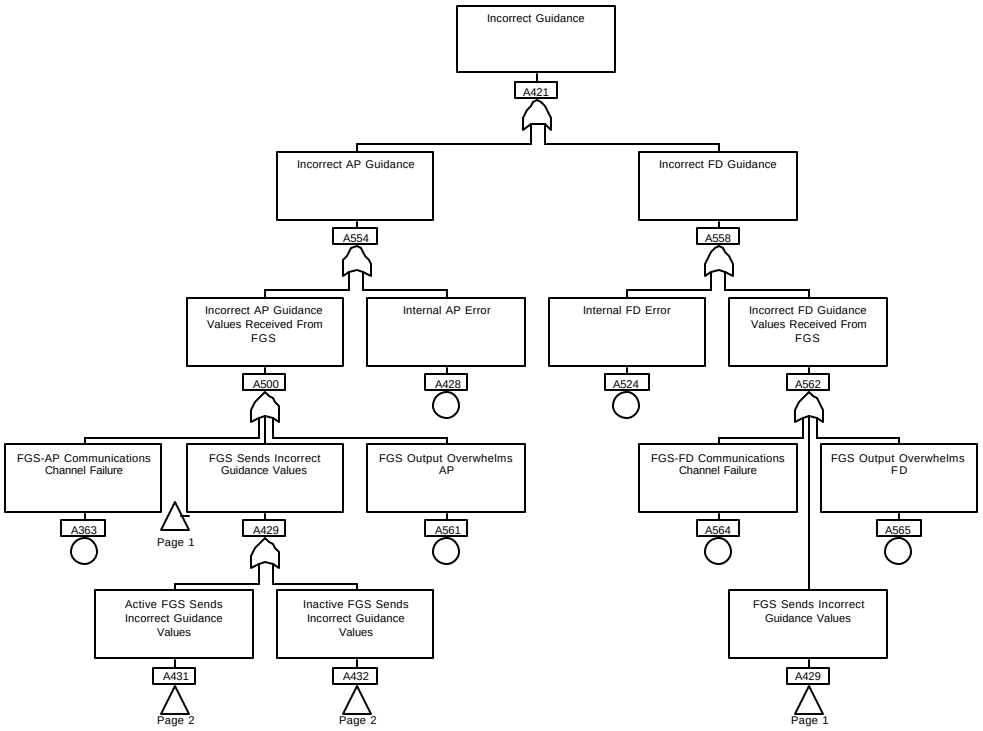

Figure 13. The Fault Tree for the Hazard – Incorrect Guidance: Part 1.

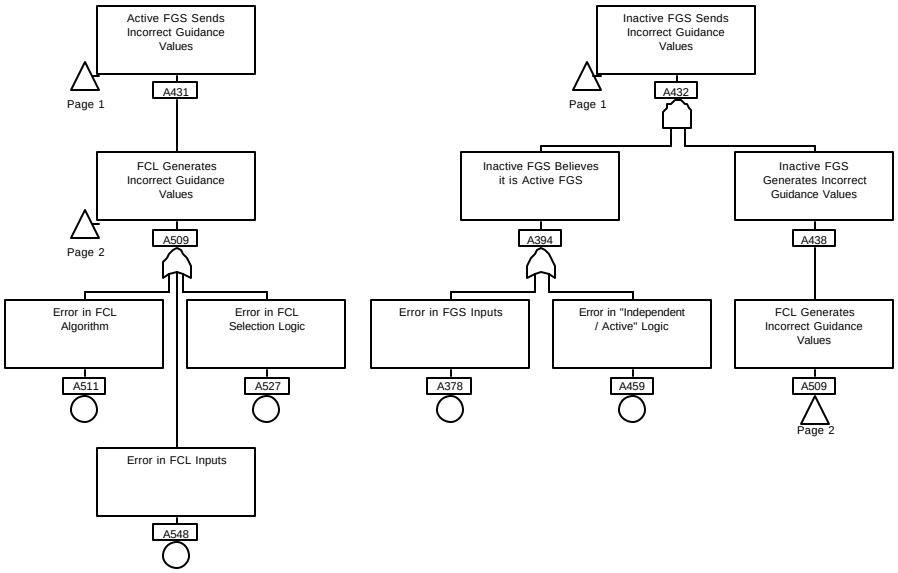

Figure 14. The Fault Tree for the Hazard – Incorrect Guidance: Part 2.

The active FGS could send incorrect guidance values due to: "Error in the FCL Algorithm" – a mistake in the mathematical relation; "Error in FCL Inputs" – incorrect input produces incorrect output; or "Error in FCL Selection Logic" – data was sent to (received from) the wrong FCL.

Any of these preceding events could cause the inactive FGS to generate the incorrect values as well, but the inactive FGS would not send these values unless it also believed it was active. This could occur due to: "Error in FGS Inputs" – misleading the FGS into believing it was active; or "Error in Independent / Active Logic" – where the wrong side is activated. For completeness, the remaining FTA's are shown in Appendix D.

As is summarized in Table 12 and Table 13, the FTA has identified twenty-three (23) separate base events that could generate one of the four major hazards associated with FGS functional failures. Upon inspection, it is seen that many of the base events that could initiate these hazards are due to failures outside of the FGS software itself. For example, the FGS output may be faulty because it stemmed from input that was corrupted before it entered the FGS, (e.g., Error in FGS Inputs). Alternatively, the FGS output may have been correct when it left the FGS but was corrupted before it was received by another entity, (e.g., FGS-FCP Communications Channel Failure). On an actual program all events are addressed by the comprehensive system level safety analysis. However for our purposes we will drop from future consideration the sixteen (16) base events that are clearly associated with functional failures outside of the FGS, Table 12. We will carry the seven (7) remaining FGS centric base events forward into the next stage of the analysis, Table 13.

It should be noted that one category of events noted in Table 11 was not seen to trace to a major level hazard in the FTA's. In particular, the category "Error in FD Selection Logic" was not identified in any of the four FTA's. This is an indication that a failure in that area would trace to at most a Minor level hazard.

Table 12. The Non-FGS Software Base Events Identified in the Fault Tree Analysis.

		Hazard			
Ref.	**Base Event**	**Guid.**	**Mode Ind.**	**Txfr State**	**AP Eng.**
A428	Internal AP Error	Yes	-	-	-
A361	Internal FCP Error	-	Yes	-	-
A524	Internal FD Error	Yes	-	-	-
A362	Internal PFD Error	-	Yes	Yes	Yes
A561	FGS Overwhelms AP	Yes	-	-	-
A366	FGS Overwhelms FCP	-	Yes	-	-
A565	FGS Overwhelms FD	Yes	-	-	-
A367	FGS Overwhelms PFD	-	Yes	Yes	Yes
A363	FGS-AP Communications Channel Failure	Yes	-	-	-
A456	FGS-FCP Communications Channel Failure	-	Yes	-	-
A564	FGS-FD Communications Channel Failure	Yes	-	-	-
A546	FGS-PFD Communications Channel Failure	-	Yes	Yes	Yes
A378	Error in FGS Inputs	Yes	Yes	Yes	Yes
A511	Error in FCL Algorithm	Yes	-	-	-
A548	Error in FCL Inputs	Yes	-	-	-
A527	Error in FCL Selection	Yes	-	-	-

Table 13. The FGS Software Base Events Identified in the Fault Tree Analysis.

		Hazard			
Ref.	**Base Event**	**Guid.**	**Mode Ind.**	**Txfr State**	**AP Eng.**
A556	Error in Annunciation Logic	-	Yes	-	-
-	Error in FD Selection Logic	-	-	-	-
A571	Error in Pilot Flying Transfer Logic	-	-	Yes	-
A459	Error in Independent / Active Logic	Yes	Yes	-	-
A567	Error in AP Engagement Logic	-	-	-	Yes
A569	Error in Mode Selection Logic	-	Yes	-	-
A521	Error in Synchronization Logic	-	Yes	-	-

4.3 Failure Mode Effects Analysis (FMEA)

The first step in the FMEA is to develop a listing of the possible failure modes. These failure modes are not simply the top level FGS functional failures identified by the FHA, but rather the specific subsystem level failures identified in Table 11. Each of these subsystem failures are the modes through which the functional failures may arise. Each of these failure modes is mapped into its worse case effect (hazard), as shown in the tables that follow.

Table 14 presents the FMEA for the failure mode – error in annunciation logic. This is a failure in the determination of when the PFD mode annunciation values, or FCP lamp illumination commands, should be output. As is seen, this failure mode could result in the loss of mode indication or an incorrect mode indication..

Table 14. The FMEA for the Failure Mode: Error in Annunciation Logic.

Effect	Analysis
Loss of Mode Indication	Failing to map mode indication data to output.
Incorrect Mode Indication	Mapping mode indication data improperly.

Table 15 presents the FMEA for the failure mode – error in FD selection logic. If the FD selection logic were faulty it could make it impossible to activate or deactivate the FD, two minor hazards. Similarly, the logic may recognize the wrong data values as commands to activate or de-activate the FD. However, these are also minor hazards.

Table 15. The FMEA for the Failure Mode: Error in FD Selection Logic.

Effect	Analysis
Unable to Activate FD	Possible if logic never recognizes request.
Inadvertent FD Activation	Possible if logic recognizes spurious request as activation command.
Unable to De-Activate FD	Possible of logic never recognizes request.
Inadvertent FD De-Activation	Possible if logic recognizes spurious request as de-activation command.

Table 16 presents the FMEA for the failure mode – error in pilot flying transfer state logic. If the transfer state logic were faulty the FGS could recognize the incorrect side as the PF side, which could in turn reverse the active / inactive sides. This could possibly result in guidance or mode indications other than were anticipated if NAV mode were active and the two flight crew members had selected different navigation sources. However, if this is annunciated clearly to the flight crew it is not a hazard. The only major hazard resulting from this failure mode is the incorrect transfer state indication, which could occur if the logic reversed the proper designation of the PF and PNF sides.

Table 16. The FMEA for the Failure Mode: Error in Pilot Flying Transfer State Logic.

Effect	Analysis
Loss of Transfer Control of Guidance Data to AP	Failure to recognize PF transfer command.
Inadvertent Transfer of Guidance Data to AP	Possible if logic recognizes spurious commands.

Table 17 presents the FMEA for the failure mode – error in independent / active logic. This is the logic element that determines whether the FGS is active or inactive. Were this logic faulty, it could result in a scenario where the FGS was inactive when it should have been active or vice versa. As with the previous FMEA, this could create a condition where the active FGS did not respond to FCP commands to change modes because it believes it is inactive. This could in turn give rise to incorrect guidance or an incorrect mode indication.

Table 17. The FMEA for the Failure Mode: Error in Independent / Active Logic.

Effect	Analysis
Loss of Guidance	Possible if both FGS' believe they are inactive.
Incorrect Guidance	Possible if both FGS' believe they are active, and in NAV mode and have different navigation sources.
Loss of Mode Indication	Possible if both FGS' believe they are inactive.
Incorrect Mode Indication	Possible if both FGS' believe they are inactive.

Table 18 presents the FMEA for the failure mode – error in AP engagement logic. If the AP engagement logic were faulty it could make it impossible to engage or disengage the AP, or it could inadvertently engage or disengage the AP. All of these are minor hazards. The two possible major hazards that could result are the loss of AP engagement indication, which could occur if the logic fails to generate an engagement indication or stops generating an engagement indication due to a spurious command. The hazard incorrect AP engagement indication is more likely, if the logic reversed the definition of engaged and disengaged.

Table 18. The FMEA for the Failure Mode: Error in AP Engagement Logic.

Effect	Analysis
Unable to Engage AP	Possible if software never recognizes validity or request.
Unable to Disengage AP	Possible if software never recognizes request, but hardware backups provide override ability.
Inadvertent AP Engagement	Possible if logic recognizes spurious command as request, and also valid.
Inadvertent AP Disengagement	Possible if logic recognizes spurious command as request, or invalid.

Table 19 presents the FMEA for the failure mode – error in mode selection logic. Were the mode selection logic faulty the FGS could fail to select a mode, which could also result in loss of guidance, or could select the incorrect mode under the circumstances. The hazard incorrect guidance assumes a mismatch between the active mode and the mode that is annunciated. It should be noted that the majority of the FGS specification is the definition of the mode selection logic itself. That is, the mode selection logic is the primary functional responsibility of the FGS. As a result, the error in mode selection logic failure mode is seen to generate only three minor level hazards. This confirms two important points. First, not every functional requirement maps directly into a corresponding safety requirement, and second, choosing the system architecture properly can simplify the resulting safety analysis.

Table 19. The FMEA for the Failure Mode: Error in Mode Selection Logic.

Effect	Analysis
Loss of Guidance	Possible if no mode is selected.
Failure to Select Mode	Possible if no mode is selected.
Incorrect Mode Selection	Possible if incorrect mode is selected.

Table 20 presents the FMEA for the failure mode – error in cross channel synchronization logic. This is the logic element that synchronizes the mode of the inactive (slave) FGS to the mode of the active (master) FGS. Were this logic faulty, it could synchronize the active FGS to the inactive side and possibly create a condition where the active FGS did not respond to FCP commands to change modes because it believes it is inactive. This could in turn give rise to incorrect guidance or an incorrect mode indication.

Table 20. The FMEA for the Failure Mode: Error in Cross Channel Synchronization Logic.

Effect	Analysis
Loss of Guidance	Possible if both FGS' believe they are inactive.
Incorrect Guidance	Possible if both FGS' believe they are active, and in NAV mode and have different navigation sources.
Loss of Mode Indication	Possible if both FGS' believe they are inactive.
Incorrect Mode Indication	Possible if both FGS' believe they are inactive.

As a final check, we compared the results of the FMEA to the results of the FTA to determine if they were self-consistent. Upon the first pass we identified a few inconsistencies, which forced us to re-examine our assumptions and analysis. After a few iterations the results were consistent and we were confident that we could proceed to the next stage of the analysis.

4.4 Safety Properties

Once the FTA and the FMEA had been completed, we used the listing of base events / failure modes as the general categories of safety properties that were associated with the FGS. As the FHA showed, some categories were more safety critical than others, with four categories being associated with both major and minor level hazards while the remaining four categories being associated with only minor level hazards. Regardless, we examined the FGS requirements model for the specific instances of those properties that should be present in the model. As summarized in Table 21, we have identified 293 specific safety properties that - if violated - could contribute to one of the hazards identified FHA. Also, note that not every property can contribute to a major level hazard, while other properties may contribute to more than one major hazard. These safety properties, (examples of which are provided in Appendix E), form the starting point for the formal methods analysis that is the subject of Chapter 5.

Table 21. A Summary of the Safety Properties Identified for the FGS Model.

			# of Properties Contributing to a Major Hazard			
		# of Safety Properties	Incorrect Guidance	Incorrect Mode Indication	Incorrect Indication of Flight Transfer	Incorrect AP Engagement Indication
Safety Property Category	Error in Annunciation Logic	41	-	9	-	-
	Error in FD Selection Logic	13	-	-	-	-
	Error in Pilot Flying Transfer Logic	8	-	-	4	-
	Error in Independent / Active Logic	5	5	5	-	-
	Error in AP Engagement Logic	10	-	-	-	4
	Error in Mode Selection Logic	166	-	104	-	-
	Error in Synchronization Logic	50	-	23	-	-
	Total # of Properties	**293**	**5**	**141**	**4**	**10**

4.5 Lessons Learned

Functional Hazard Assessment (FHA)

We confirmed that a safety analysis should start with the system level hazards, which in turn stem from functional failures. Without a firm understanding of what the system is supposed to do, its functional requirements, it is impossible to fully understand the implications of the functional failures. These failures determine what hazardous conditions, if any, arise when the failure occurs. The severity of the most critical functional failure in turn determines the level of criticality for the system. The challenge to this stage was to clearly articulate the general functions required of the FGS and to understand how failure to provide this functionality would manifest itself.

Fault Tree Analysis (FTA)

We came to appreciate that a meaningful FTA must begin with a thorough understanding of the system architecture and must factor in both hardware and software issues. On the hardware side, there are two FGS units in operation at any time. Each of these may send data to one or both FD, one or both PFD's, and / or the AP, and either one may be active or inactive depending on the location of pilot flying transfer switch and the mode of operation. On the software side, the FGS logic is functionally decomposed into different elements having unique responsibilities. Understanding the functional abilities, and limitations, of both the hardware and software elements is necessary in order to develop a meaningful FTA. In other words, doing a FTA correctly is a bit of an art form. A pencil in the hands of a an experienced artist may produce a masterpiece, while the same pencil in the hands of a 5-year old would surely not. The FTA is a tool. In the hands of an experienced user the FTA may add value, while the FTA generated by a disconnected or inexperienced user could generate worthless, or misleading, information. For this reason it is important that the FTA efforts be verified by an ongoing review of the results, both with people who are familiar with the system and those who are familiar with the process. These may be very distinct sets of individuals.

Having said this, we were impressed with the power of a simple technique like FTA. Once completed, the FTA provides an easy means of illustrating the effect of errors / faults / failures and to identify whether these events are single point failures or require additional events to occur before progressing into hazards. In traditional hardware analysis a FTA would include failure probabilities so that the reliability of the entire system could be quantified. Because software does not fail like hardware we feel that including reliability numbers are not appropriate for an analysis of this nature. Software will generate the same output every time, given the same initial state and input values. However, it can fail to perform as intended if it is not properly designed. Consequently, our analysis was focused on identifying the properties of the system that relate to safety. We felt that the FTA was very effective at identifying the base events (errors) that could initiate hazards.

Failure Mode Effects Analysis (FMEA)

The FMEA is an appropriate tool for documenting the effects of failures. It is a bottom-up approach that is designed to articulate the effect of some function not being provided. We found the FMEA to be an easier tool to use than FTA, because in this situation the failure was already known. Once the initiating event was defined it was straightforward to guess at what the consequences of that failure could be.

One important point is that we feel it is necessary to close the loop, by comparing the results of the FTA to those of the FMEA and ensuring that they are self-consistent. This is often termed a Bi-Directional Analysis (BDA). Even if BDA does not identify any elements that have been overlooked, it is quite valuable in that it forces consistency. As such, it did give a much higher level of confidence in the final results. It was helpful to walk through the entire loop several times to make certain that nothing had been over looked.

Safety Properties

Once the base events were identified it was necessary to articulate the specific instances of these events that should be manifested in our requirements model. In most cases, these were simply the individual requirements statements themselves. That is, the requirements that trace directly to safety became the individual safety properties for the model. At this point it was clear that an in depth understanding of the system was required in order to ensure that all properties had been identified. That is, it is important to ensure that all the properties are complete and consistent. As will be seen in the next chapter, the formal methods analysis tools can provide this assurance.

5 Formal Methods Analysis Results

In the initial stages of our formal analysis we examined the use of both the NuSMV model checker and the PVS theorem prover for verifying the safety properties identified in the last chapter. It became evident early on that the model checker would be capable of verifying all of the safety and functional properties identified. At the same time, early experiments suggested that analyzing the FGS models for potential sources of mode confusion would require the flexibility of a theorem prover such as PVS. As a result, we made a decision to focus our model checking analysis on the FGS safety properties and use the PVS theorem prover to search for potential sources of mode confusion. This chapter will summarize the results of our analysis using the NuSMV model checker. The results with the PVS theorem are discussed in a companion paper, [29].

To optimize our progress, we approached the use of formal methods in an incremental fashion. The full model of the FGS has five lateral modes and seven vertical modes, plus a flight director, Auto-Pilot, cross-channel mode synchronization logic, and so on. We felt it would be more productive to develop techniques and strategies on smaller models and gradually build up to the full FGS. Consequently, we defined a series of models ranging in complexity from only a Flight Director and two lateral modes to the full FGS model, Table 22.

We began by translating the Level 0 model into SMV, along with a subset of the properties associated with it, and modified the model or proofs as necessary until all properties were verified as true. Once the entire subset of proofs had been completed we moved to the next model, Level 1, which added two vertical modes. We then re-ran the entire subset of proofs for the Level 0 model, before adding the additional properties associated with the two vertical modes. In this manner, we worked up to the complexity of the full FGS at Level 5. In the next phase of the project, we will extend the model even further to integrate with the FMS VNAV model being developed separately.

Because the translation of the FGS requirements model into SMV had been automated by the University of Minnesota, the only remaining challenge to the use of the model checker was the translation of the safety properties into the SMV language. Once this was completed the model checker could verify the presence of the properties in the model. Each of these stages is examined in the sections that follow, while the lessons learned from conducting the analysis are summarized in the final section.

Table 22. Incremental Approach to Model Checking.

Model	Lateral Modes Added	Vertical Modes Added	Other Capability Added	Properties Verified
Level 0	Roll Hold Heading Select	-	Flight Director	29
Level 1	-	Pitch Hold Vertical Speed	-	47
Level 2	-	Altitude Hold	Auto-Pilot Pilot Flying	76
Level 3	-	-	Cross-Channel Mode Synchronization	122
Level 4	Navigation	Altitude Select Flight Level Change	Air Data Computer Navigation References	181
Level 5 (Full Model)	Approach Go Around	Approach Go Around	Independent Operation	281

5.1 Translation of Safety Properties into SMV

The output from the traditional safety analysis was a listing of specific safety properties required of the FGS. These properties were generated as English prose and had to be translated manually into the SMV language in order for the model checker to perform its analysis. As a check on the accuracy of the translation, two investigators translated each property independently and then compared results. The translation is straightforward, but requires some knowledge of temporal logic. We briefly experimented with the possibility of expressing these properties in the language of the requirements model itself, so that the properties would be translated along with the model, but it became apparent that without extensions to the RSML^{-e} language this would involve more effort than simply translating the properties manually into SMV.

To ensure traceability of requirements and to facilitate technology transfer to the Rockwell Collins product areas we utilized the Rockwell Collins standard requirements management tool, the Dynamic Object Oriented Requirements System (DOORS), as the repository for the original English prose statements of the FGS requirements. DOORS, by Telelogic, is a popular commercial tool for managing requirements. It is currently used by more than 50,000 users at over 1,000 companies around the world and provides a variety of capabilities to capture and link information to ensure compliance with specified requirements.

As a first step, we created a requirements document in DOORS for the English statements of the FGS requirements. We then added within DOORS a corresponding statement in the syntax of SMV, Figure 15. This provided complete traceability for all requirements. While most of these requirements related to safety, some were purely functional in nature. That is, some of the

properties did not trace directly to a base event identified in the traditional safety analysis. However, model checking worked equally well for verifying that both functional and safety properties were met.

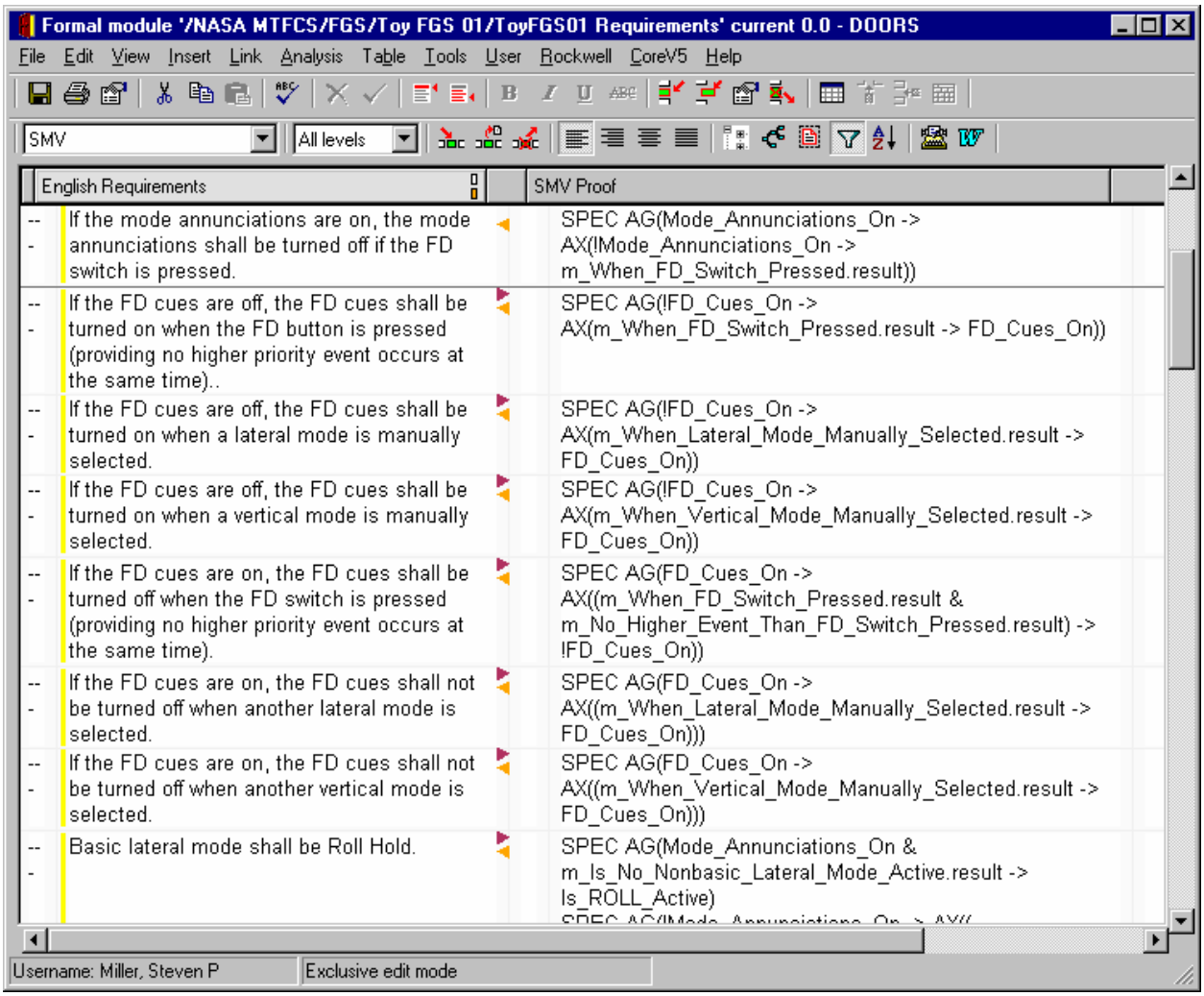

Figure 15. DOORS was Used to Capture the Requirements in Both English and SMV

Once the properties, in both English and SMV, were captured in DOORS we created a simple script to generate a text file that listed the English statement, (preceded by the SMV comment delimiter so that it did not interfere with the operation of the tool), followed by the SMV equivalent. In this way, the properties could be exported directly into the SMV model from DOORS within a few seconds.

All of the requirements could be translated into one of two formats. The first format was simply a constraint that had to be maintained on all reachable states. For example, the requirement

> *If this side is active, the mode annunciations shall be on if and only if the onside FD cues are displayed, or the offside FD cues are displayed, or the AP is engaged.*

was translated into the SMV property

```
SPEC AG(Is_This_Side_Active = 1  ->
                (Mode_Annunciations_On <->
                (Onside_FD_On | Offside_FD_On = TRUE | Is_AP_Engaged)))
```

where the AG operator states that the property must hold for all globally reachable states and the operators -> and <-> have their usual meaning of "implies" and "iff".

The second format was a constraint over a state and all possible next states. For example, the requirement

> *If the onside FD cues are off and the AP is not engaged, the onside FD cues shall be displayed when the AP is engaged.*

was translated into the SMV property

```
SPEC AG((!Onside_FD_On & !Is_AP_Engaged) -> AX(Is_AP_Engaged -> Onside_FD_On))
```

Where the AX operator states the enclosed property must hold for all states reachable in the next step.

These two formats were sufficient because RSML^{-e} is a synchronous language in which the transition from each state to the next state is computed in a single atomic transition. If portions of the model had been allowed to evolve asynchronously, then other temporal logic operators such as eventually (F), until (U), or release (R) would have been required, [16].

Whenever possible, we tried to formulate the SMV properties in terms of the monitored (MON') and controlled (CON') variables identified in the RSML^{-e} specification (see the discussion of the four-variable model in Section 2.4). In other words, the properties verified that the model correctly implemented the **REQ'** relation. This made the properties more independent of the internal organization of the model and a verification of the "end-to-end" behavior of the model. Sometimes, this proved impractical and properties were stated in terms of macros or internal state variables of the model. For example, for some properties, it was necessary to refer to the RSML^{-e} macro *When Lateral Mode Manually Selected*, which was in turn defined in terms of the monitored variables. Stating each property in term of the expansion of this macro would have been tedious and prone to error. However, this required a separate set of properties to ensure the macro itself was correctly defined.

It would have been possible to define properties in terms of the input and output fields defined in the RSML^{-e} interfaces rather than the monitored and controlled variables. In terms of the four variable model, the properties would have verified the composition of the **IN'**, **REQ'**, and **OUT'** relations as discussed in Section 2.4. However, this would have made the properties dependent on the input and output fields in the interfaces. We felt this was undesirable because we were most interested in verifying that **REQ** was correctly specified. It also would have made the properties more fragile in that the input and output variables were more likely to change than the monitored and controlled variables.

For this reason, we broke the properties down into three sets. The first set verified the **REQ** relation as described above. The second set verified the **IN'** relation described in the model, i.e., they verified that the monitored variables were correctly constructed from the input variables. The third set verified the **OUT'** relation, i.e., they verified that the output variables were correctly constructed from the controlled variables.

While the SMV translations may appear somewhat arcane, the process quickly became routine and straightforward. Representative examples of the requirements document, in both English prose and SMV syntax, are included in Appendix E.

5.2 Running the Proofs

Once the SMV model had been generated from the requirements model, and the properties were available in a text file, running the proofs was simply a matter of inserting the properties into the SMV model and executing the state space search. Originally, the proofs for the Level 0, 1, and 2 models executed in a matter of seconds, while the proofs for the Level 3 model executed in a little more than a minute. However, the total state space of the Level 4 model was large enough that the proofs ran for several hours. Since it was clear that the limits of the NuSMV model checker would be reached before the Level 5 model was completed, the University of Minnesota improved the RSML^{-e} to SMV translator to optimize the proof process. These changes included improvements to the translation of RSML^{-e} macros to avoid the introduction of redundant state information into the SMV model and the implementation of interface abstraction that enables selective translation of the input and output interfaces. These enhancements, especially the improvements to the translation of RSML^{-e} macros, enabled the model checker to verify the 181 properties for the Level 4 model in about four minutes. The 293 properties for the Level 5 model required about an hour to verify.

While a turn around time of a few hours is reasonable for a completed product, it is still too long to enable efficient debugging of prototypes. As a result, we used a variety of simple techniques to reduce the size of the model during the initial stages of verification. For example, it was easy to separate the lateral modes from the vertical modes and verify their properties separately. In a similar fashion, eliminating the low level inputs and outputs from the model (i.e., those portions of the model corresponding to the **IN** and **OUT** relations of the four-variable model described in Section 2.4), also speeded up the process. We found that for the initial stages of the model checking, when most of the counter-examples were found, it was most useful to reduce the model in this way so that the errors could be found in seconds rather than minutes or hours. Of course, all properties were later verified in the full model to ensure that no unforeseen interactions between these components had been introduced.

5.3 Lessons Learned

Model Based Development Enables Formal Methods Analysis

A significant issue in transferring formal methods into practice is the fact that most practicing engineers will not be trained in the use of formal methods tools and techniques. As such, it will be difficult to discuss results with them unless some common notation for discussing the problem

domain can be found. We were pleased to see that the RSML^{-e} language seems to offer such a common ground. With only a few minutes of training, we found that most systems designers could understand the RSML^{-e} models. The most difficult part of the formal verification was manually translating the English requirements into SMV properties, and even this could be easily mastered by most practitioners. An even better approach would be to extend the RSML^{-e} language so that properties could be specified in RSML^{-e} and automatically translated into SMV along with the model itself. However properties are specified, the selection of a domain specific language that practitioners will use and that has a well defined, formal semantics appears to be a prerequisite for the industrial use of formal methods.

Model Checking is Ready for Industrial Use

Although there are some limitations to the use of model checking that are discussed in the following pages, we were very impressed by both the ease of use of this technology and its ability to generate meaningful results when applied to the FGS mode logic. Admittedly, the FGS mode logic is admirably suited to verification through model checking. It consists of a large number of small, tightly synchronized, finite-state machines with few integer or real variables. Model checking would probably not have been as successful if the problem domain had included more real-valued variables or functions. However, there are many such applications in industrial systems that can be modeled as finite-state systems, or reduced to a finite-state system through simple abstraction techniques. Even in the FGS mode logic, we replaced some real variables and the comparisons based on them (e.g., Altitude > 18,000 ft) with a simple Boolean input. Certainly, model-checking can be recommended for problems of this nature without reservation.

Verifying properties of a requirements model was also an advantage. Requirements models tend to be smaller than design models or code. At the same time, there is considerable evidence that the most common and most serious errors in system development are requirements errors, so verification of properties of these models is likely to provide the greatest return. If code is automatically generated from the models, it becomes even more important to verify that the model exhibits the required behavior.

We did see clear evidence that the state space problem is real and limits the size of the models that can be verified through model-checking. However, there are a variety of abstraction techniques under development that should enable the verification of larger models. Other techniques, such as compositional reasoning, may make it possible to verify components of a system and then verify properties of assemblies of these components. Until these techniques are perfected, practitioners can still improve their confidence in their systems by verifying properties of the most difficult parts of their models.

Restating the Requirements in SMV Improves the Requirements

In an earlier section we discussed how the process of creating a model from the English prose requirements caused us to go back and clarify the English statement of the requirements. In the same way, translating the English statements into SMV also prompted us to go back and clarify the English statement. For example, in trying to prove the requirement

If Heading Select mode is not selected, Heading Select mode shall be selected when the HDG switch is pressed on the FCP

We discovered two ways in which this property might not be true. First, if a higher priority event arrives at the same time as the HDG switch is pressed, that event may preempt the HDG switch pressed event. Second, if this side of the FGS is not active and the active side fails to receive or process the HDG switch pressed event, the inactive side will not respond to the event. This led us to modify the requirement to state

If this side is active and Heading Select mode is not selected, Heading Select mode shall be selected when the HDG switch is pressed on the FCP (providing no higher priority event occurs at the same time)

While this requirement is longer and more difficult to read than the original statement, it has the advantage of being an accurate description of the system's behavior.

We found that the process of proving the properties forced us to go back and modify virtually all of our original English requirements. At the conclusion of this process, we were far more satisfied that our English requirements were as complete and consistent as English prose could be.

Desired Improvements to the NuSMV Tool

Two aspects of the NuSMV tool made it difficult to use. The first was that the properties had to be embedded in the model itself or fed in using a batch file while in interactive mode. Embedding the properties in the models indirectly requires the user to modify the model file any time the properties were changed, which was frequent in the early stages of verification. Adding a simple "include *filename*" statement to the NuSMV language would solve this problem. This same construct could be used to organize large models into separate files.

The second issue, which was far more time consuming, is that the counter-examples are generated in a format that is difficult to read. The counter-examples are output as a text file that lists all state variables in the initial state, and then lists all variables that have changed in each subsequent state. In an automatically generated model with several hundred variables, each with a lengthy name generated by the translator from the original RSML^{-e} name, it was almost impossible to decipher even a short counter-example. We found it most useful to reformat the information so that the initial value of all of the variables were listed in the first vertical column, and their value in each subsequent state was presented in each following vertical column. This made it much easier to understand which variables were changing values from state to state.

Care Must be Taken to Write Meaningful Properties

Care does need to be taken when formulating SMV properties to ensure that their proofs are meaningful. For example, we previously discussed how RSML^{-e} macros such as *When Lateral Mode Manually Selected* were occasionally used in stating the SMV properties. In most cases, this macro was used as the antecedent of an implication, for example,

SPEC AG(m_When_Lateral_Mode_Manually_Selected.result -> Onside_FD_On)

However, if the macro *When Lateral Mode Manually Selected* had the value false, this proof would always succeed, rendering the proof meaningless. To be meaningful, the properties should also verify that the macro itself is correctly defined. Similar issues can arise if the individual specifying the properties is unfamiliar with logic or temporal logic. For example, many software and system engineers might not fully appreciate the difference between implication (IMPLIES) and equivalence (IFF).

Model Checking Found Several Important Errors

The most important lesson learned, other than the fact that this approach is ready for industrial use, was that it is capable of identifying errors in requirements models that might have otherwise escaped detection until far later in the product life cycle. Model checking found many errors in the original English statement of the requirements and several errors in the model itself. Many of the errors in the model were minor and probably would have been found during design or implementation. However, we did discover errors that could have escaped detection until quite late in the design cycle.

One of the most common errors was failing to define the behavior of the system when more than one input event occurred at the same time. This could occur for a variety of reasons. For example, the pilot might press a switch at the same as the copilot selects a different switch. Or the pilot might press a switch at the same time as the capture of the lateral navigation source occurs. Frequently, these interactions could drive the FGS model into an unsafe state in which more than one mode was active at the same time or in which no mode was active.

Rather than trying to fix this problem by modifying the specification to handle simultaneous input events, we elected to assign a priority to the input events and only use the highest priority event in each cycle, ignoring the lower priority events. While this prioritization obviously needs to be reviewed by the domain experts for its safety implications, this has the advantage of isolating the prioritization to one location in the specification.

An added benefit was that the model-checker allowed us to confirm that partial order, rather than a total order, of the input events was acceptable, Figure 16. That is, it was acceptable for some combinations of events to occur at the same time. Without the power of formal verification, we would never have been able to convince ourselves that this was safe and would have definitely opted for only allowing one event to be processed during each cycle.

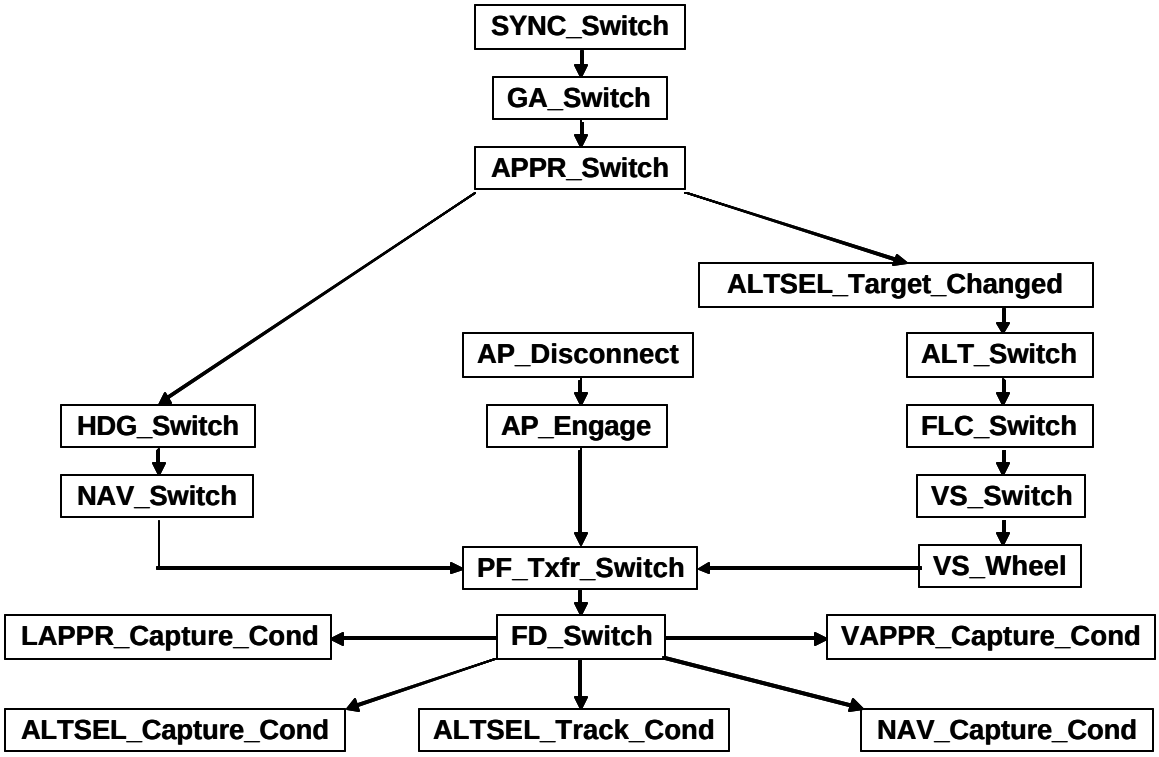

Figure 16. The Partial Ordering of Event Priorities.

6 Summary and Conclusions

We have developed a methodology for dove-tailing traditional safety analysis techniques such as Functional Hazard Assessment (FHA), Fault Tree Analysis (FTA), and Failure Mode Effects Analysis (FMEA), with formal methods to conduct a comprehensive safety analysis of a software centric avionics function, a Flight Guidance System (FGS). By using a Model Based Development (MBD) approach, we built a requirements model for the FGS in the RSML^{-e} language. We then conducted a FHA to map the FGS functional failures into system level hazards. Using a FTA we then performed a top-down analysis of the hazards to identify the events (errors / faults / failure) that could initiate them. These event categories were compared against the requirements model in order to develop a complete listing of safety properties for the requirements model. As a check, we also conducted a Failure Mode Effects Analysis (FMEA) in order to verify that our results were self-consistent.

We then translated the requirements model into the NuSMV model checker and used this formal methods tool to verify the presence of the safety properties in the requirements model itself. In particular, we have verified 293 properties of the full FGS model. Specific safety milestones that have been completed include:

- Functional Hazard Assessment (FHA);
- Fault Tree Analysis (FTA);
- Failure Mode Effects Analysis (FMEA);
- Definition of FGS Safety Properties;
- Translation of the FGS Model and all Safety Properties into the SMV Model Checker;
- Translation of the FGS Model and some Safety Properties into the PVS Theorem Prover;
- All Safety Properties Verified with the NuSMV Model Checker;
- Some Safety Properties Verified with the PVS Theorem Prover;

Our main conclusion is that model checking is ready for industrial use and can greatly augment the safety analysis for some classes of problems, specifically those with reasonable ($< 10^{20}$) state spaces. The formal methods analyses conducted to date has not only had the benefit of providing a higher level of confidence in the final model, but may ultimately prove to be an additional means of helping to certify the system. This lends credence to the belief that such approaches may become an integral part of future model based development efforts, [30].

Appendix A - Bibliography

[1] Jacobsen, R., NASA Airspace Systems Program, Virtual Airspace Modeling and Simulation Project, Technical Interchange Meeting #1, 21 May 2000.

[2] IEEE Std. 610.12-1990, Standard Glossary of Software Engineering Terminology.

[3] Parnas, D. L. and J. Madey, "Functional Documentation for Computer Systems Engineering, Vol. 2," McMaster University, Hamilton, Ontario, Technical Report CRL 237, September 1991.

[4] Heitmeyer, C. L., J. Kirby, and B. G. Labaw, "Automated Consistency Checking of Requirements Specification," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 5, No. 3, pp. 231-261, July 1996.

[5] Miller, S. P. and A. C Tribble, "Extending the Four Variable Model to Bridge the System Software Gap," 20th AIAA / IEEE Digital Avionics Systems Conference, Daytona Beach, FL, October 2001.

[6] Clarke, E. O., O. Grumberg and D. Peled, *Model Checking*, (Cambridge, MA: MIT Press, 2000).

[7] www-2.cs.cmu.edu/~modelcheck/smv.html

[8] www.nusmv.irst.itc.it

[9] Owre, S., J. Rushby, N. Shankar, and F. Henke, "Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS," *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, pp. 107-125, February 1995.

[10] Butler, R. W., "An Elementary Tutorial on Formal Specification and Verification Using PVS, NASA TM 108991, September 1993.

[11] www.pvs.csl.sri.com

[12] Leveson, N., M. Heimdahl, H. Hildreth, and J. Reese, "Requirements Specifications for Process-Control Systems," IEEE *Transactions on Software Engineering*, Vol. 20, No. 9, pp. 684-707, September 1994.

[13] Harel, D., and A. Naamad, The STATEMATE Semantics of Statecharts, *ACM Transactions on Software Engineering and Methodology*, Vol 5., No. 4, pp. 293-333, October, 1996.

[14] Leveson, N., et al., Safety Analysis of Air Traffic Control Upgrades, NASA TR, September 1997.

[15] Thompson, J. M., M.. P.E. Heimdahl, and S. P. Miller, "Specification Based Prototyping for Embedded Systems," 7th ACM Symposium on the Foundations on Software Engineering, September 1999.

[16] Heitmeyer, C. L., "Software Cost Reduction," in *Encyclopedia of Software Engineering*, J. J. Marciniak, (Ed.), January 2002.

[17] Bharadwaj, R. and C. Heitmeyer, "Developing High Assurance Avionics Systems with the SCR Requirements Method," 19th AIAA / IEEE Digital Avionics Systems Conference, Philadelphia, PA, October 2000.

[18] Caspi, P., D. Pilaud, N. Halbwachs, and J. A. Plaice, "LUSTRE: A Declarative Language for Programming Synchronous Systems," 14th ACM Symposium on Principles of Programming Languages (POPL), pp. 178 – 188, 21 – 23 January 1987.

[19] Halbwachs, N., P. Caspi, P. Raymond, and D. Pilaud, "The Synchronous Dataflow Programming Language LUSTRE," *Proc. IEEE*, Vol. 79, No. 9, pp.1305 - 1320, 1991.

[20] Berry, G. and G. Gonthier, "The Synchronous Programming Language ESTEREL: Design, Semantics, and Implementation," *Science of Computer Programming*, Vol. 19, pp. 87-152, 1992.

[21] Boussinot, F. and R. De Simone, "The ESTEREL Language," *Proc. IEEE*, Vol. 79, No. 9, pp. 1293 - 1304, 1991.

[22] ARP 4754, "Certification Considerations for Highly-Integrated or Complex Aircraft Systems," SAE International, November 1996.

[23] ARP 4761, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," SAE International, December 1996.

[24] RTCA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 01 DEC 1992.

[25] MIL STD 882C, System Safety Program Requirements, 19 January 1993.

[26] *System Safety Analysis Handbook*, 2nd Ed., System Safety Society, July 1997.

[27] Miller, S. P., "FGS Final Report," Delivered to NASA LaRC as Part of NCC-01-001 on 30 November 2001.

[28] Heimdahl, M. P. E., "Proof and Model Checking Tools Final Report," Delivered to NASA LaRC as Part of NCC-01-001 on 30 November 2002.

[29] Miller, S. P., "FGS Mode Confusion Final Report," Delivered to NASA LaRC as Part of NCC-01-001 on 30 November 2002.

[30] Tribble, A. C., "Software Safety," *IEEE Software*, pp. 84 - 85, June – July, 2002.

Appendix B - Acronyms

ADS	Air Data System
AHRS	Attitude, Heading, and Reference System
ALT	Altitude
ALTS	Altitude Select
AP	Auto-Pilot
APPR	Approach
AT	Auto-Throttle
BDD	Binary Decision Diagram
DCP	Display Control Panel
FCS	Flight Control System
FD	Flight Director
FCP	Flight Control Panel
FGS	Flight Guidance System
FHA	Functional Hazard Assessment
FLC	Flight Level Change
FMEA	Failure Modes Effects Analysis
FMS	Flight Management System
FTA	Fault Tree Analysis
GA	Go Around
GS	Glide Slope
HDG	Heading
IAS	Indicated Air Speed
LOC	Localizer
MBD	Model Based Development
NAV	Navigation
PF	Pilot Flying
PFD	Primary Flight Display
PNF	Pilot Not Flying
PSA	PreSelect Altitude
PTCH	Pitch
ROLL	Roll
SYNC	Synchronize
VOR	VHF Omnidirectional Ranging
VNAV	Vertical Navigation
VS	Vertical Speed

Appendix C - Definitions

Accident	An unplanned event, or series of events, that results in death, injury, illness, environmental damage, or damage to or loss of equipment or property.
Defect	An error that makes it into operation.
Error	A mistake in requirements, design, or implementation. (Before Operation)
Error Detection and Correction	The processes used to find and fix errors.
Fail-Safe	The design features that ensures a system remains safe, or in the event of failure, will cause the system to revert to a state that will not cause an accident.
Failure	The inability of a system or component to perform its required functions within specified performance requirements.
Fault	The manifestation of an error, or defect, during operation.
Fault Tolerance	The ability of a system or component to continue normal operation despite the presence of hardware or software faults.
Hazard	Any real or potential condition of a system that, together with other conditions in the environment of the system, will lead to an accident.
Incident	The occurrence of a hazardous condition.
Reliability	The ability of a system or component to perfo rm its required functions under stated conditions for a specified period of time.
Safety	Freedom from the conditions that cause accidents.

Appendix D - Fault Tree Analysis Results

The FTA for the hazard, *Incorrect Mode Indication*, is shown in Figure 17 and Figure 18. Note that the FCP receives lamp illumination commands separately from each FGS, so there is no dependence on whether the FGS is active or inactive. In contrast, each PFD receives the same mode command from the active FGS so its fault tree must address the possibility of the inactive FGS sending incorrect mode annunciations. The event "Error in Annunciation Logic" is included in each tree. Other contributing factors include "Error in Synchronization Logic", "Error in Mode Selection Logic", and "Error in Independent / Active Logic".

The FTA for the hazard, *Incorrect Transfer State Indication*, is shown in Figure 19 and Figure 20. Here the contributing factors include "Error in Annunciation Logic", "Error in Pilot Flying Transfer Logic", "Error in Independent / Active Logic", and "Error in FGS Inputs".

The FTA for the hazard, *Incorrect AP Engagement State Indication*, is shown in Figure 21 and Figure 22. As with the preceding fault trees, contributions are seen to "Error in Annunciation Logic", "Error in Independent / Active Logic", "Error in AP Engagement Logic", and "Error in FGS Inputs".

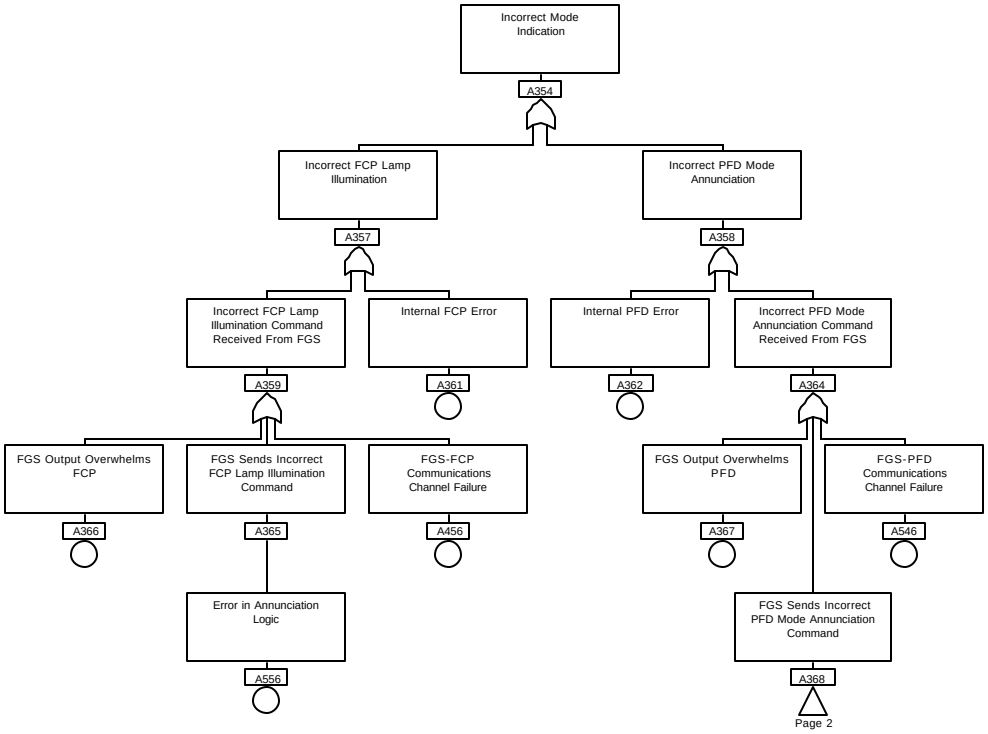

Figure 17. The Fault Tree for the Hazard – Incorrect Mode Indication: Part 1.

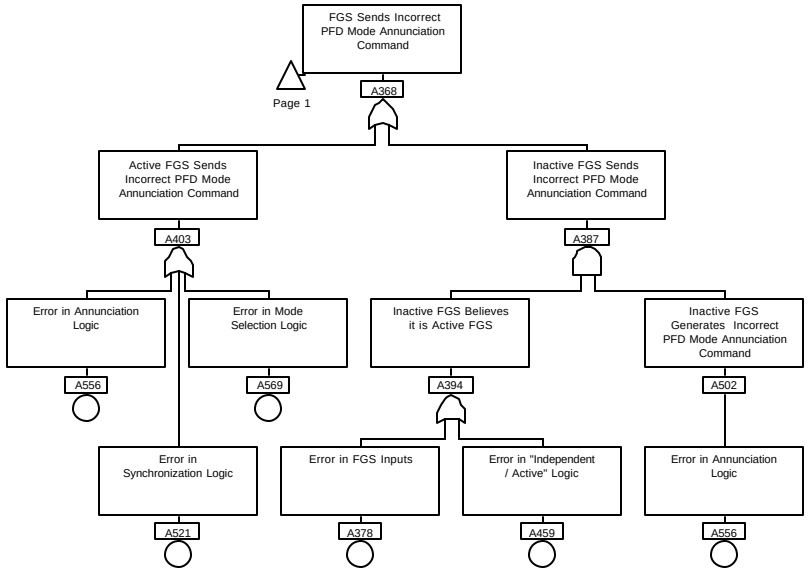

Figure 18. The Fault Tree for the Hazard – Incorrect Mode Indication: Part 2.

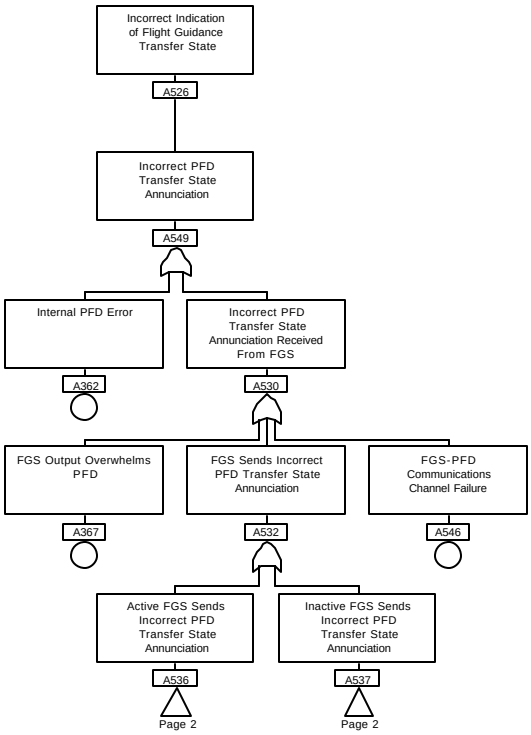

Figure 19. The Fault Tree for the Hazard – Incorrect Transfer State Indication: Part 1.

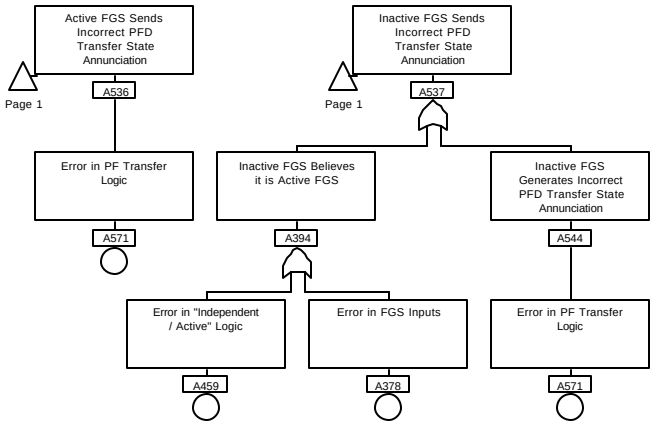

Figure 20. The Fault Tree for the Hazard – Incorrect Transfer State Indication: Part 2.

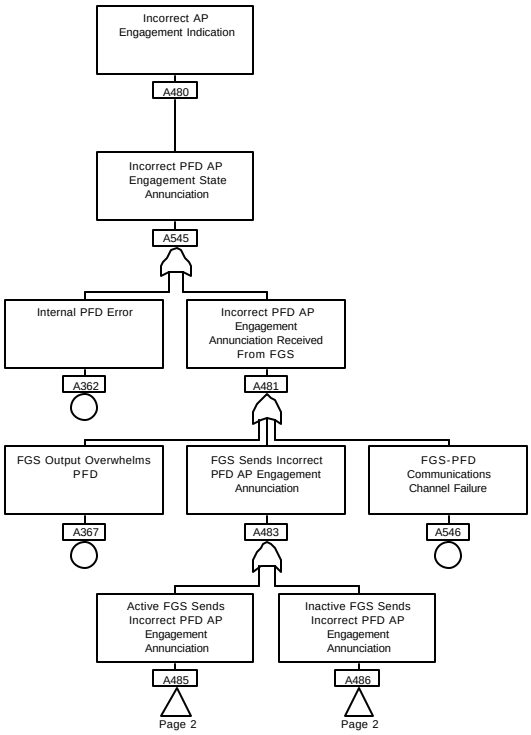

Figure 21. The Fault Tree for the Hazard – Incorrect AP Engagement Indication: Part 1.

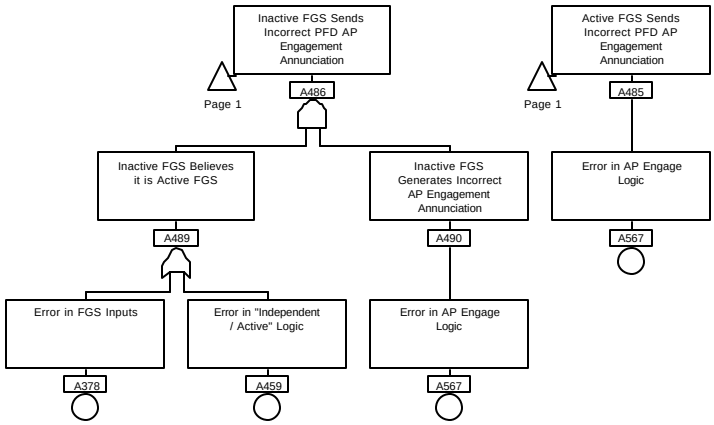

Figure 22. The Fault Tree for the Hazard – Incorrect AP Engagement Indication: Part 2.

Appendix E - FGS Requirements / Properties

The following pages provide excerpts from the requirements specification for the FGS model. The information is presented first in English prose and then in the SMV syntax. The appendix is therefore an illustration of the type of properties that were verified using the SMV model checker. The outline of the appendix is as follows.

1. Annunciations
2. Flight Director
3. Pilot Flying Transfer
4. Independent & Active
5. Auto-Pilot Engagement
6. Mode Selection
   - 6.1 Lateral Modes
     - 6.1.1 Operation
     - 6.1.2 Roll (ROLL)
     - 6.1.3 Heading Select (HDG)
     - 6.1.4 Navigation (NAV)
     - 6.1.5 Lateral Approach (LAPPR)
     - 6.1.6 Go Around (GA)
   - 6.2 Vertical Modes
     - 6.2.1 Operation
     - 6.2.2 Pitch (PTCH)
     - 6.2.3 Vertical Speed (VS)
     - 6.2.4 Altitude Hold (ALT)
     - 6.2.5 Altitude Select (ALTSEL)
     - 6.2.6 Flight Level Change (FLC)
     - 6.2.7 Vertical Approach (VAPPR)
     - 6.2.8 Go Around (GA)
7. Cross Channel Synchronization

1. Annunciations

1.1 Selection

English

If this side is active and the mode annunciations are off, the mode annunciations shall be turned on when the onside FD is turned on.

SMV

SPEC AG((!Mode_Annunciations_On & !Onside_FD_On) -> AX((Is_This_Side_Active = 1 & Onside_FD_On) -> Mode_Annunciations_On))

1.2 De-Selection

English

If this side is active and the mode annunciations are on, the mode annunciations shall be turned off if the onside FD is off, the offside FD is off, and the AP is disengaged.

SMV

SPEC AG(Mode_Annunciations_On -> AX((Is_This_Side_Active = 1 & !Onside_FD_On & Offside_FD_On = FALSE & !Is_AP_Engaged) -> !Mode_Annunciations_On))

1.3 Operation

English

The mode annunciations shall not be on at system power up.

SMV

SPEC (!Mode_Annunciations_On)

2. Flight Director

2.1 Selection

English

If the onside FD cues are off, the onside FD cues shall be turned on when the FD button is pressed (providing no higher priority event occurs at the same time).

SMV

SPEC AG(!Onside_FD_On -> AX((m_When_FD_Switch_Pressed.result & m_No_Higher_Event_Than_FD_Switch_Pressed.result)-> Onside_FD_On))

2.2 De-Selection

English

If the onside FD cues are on, the onside FD cues shall be turned off when the FD switch is pressed (providing no higher priority event occurs at the same time).

SMV

SPEC AG(Onside_FD_On -> AX((m_When_FD_Switch_Pressed.result & m_No_Higher_Event_Than_FD_Switch_Pressed.result & Overspeed = 0) -> !Onside_FD_On))

2.3 Operation

English

If the onside FD cues are on, the onside FD cues shall not be turned off when another lateral mode is manually selected.

SMV

SPEC AG(Onside_FD_On -> AX((m_When_Lateral_Mode_Manually_Selected.result -> Onside_FD_On)))

3. Pilot Flying Transfer

3.1 Selection

English

The PF shall be transferred to the other side when the PF Transfer switch is pressed (providing no higher priority event occurs at the same time).

SMV

SPEC AG(Pilot_Flying=LEFT -> AX((m_When_Transfer_Switch_Pressed.result & m_No_Higher_Event_Than_Transfer_Switch_Pressed.result) -> Pilot_Flying=RIGHT))

SPEC AG(Pilot_Flying=RIGHT -> AX((m_When_Transfer_Switch_Pressed.result & m_No_Higher_Event_Than_Transfer_Switch_Pressed.result) -> Pilot_Flying=LEFT))

3.2 Operation

English

If the mode annunciations are on, changing the PF side shall cause Roll Hold mode to become the active lateral mode (providing this side becomes active and no higher priority event occurs at the same time and an overspeed condition does not exist).

SMV

SPEC AG(Mode_Annunciations_On -> AX((Is_This_Side_Active = 1 & Mode_Annunciations_On & m_When_Transfer_Switch_Pressed.result & m_No_Higher_Event_Than_Transfer_Switch_Pressed.result) -> Is_ROLL_Active))

4. Independent / Active

4.1 Independent / Dependent

English

The independent mode condition will be met whenever LAPPR and VAPPR are active on both the onside and offside FGS.

SMV

SPEC AG((Is_LAPPR_Active & Is_VAPPR_Active & Offside_Lappr_Selected = 1 & Offside_Vappr_Selected = 1) -> m_Independent_Mode_Condition.result)

4.2 Active / Inactive

English

This side shall be active if the independent mode conditions are met.

SMV

SPEC AG(m_Independent_Mode_Condition.result <-> Is_This_Side_Active = 1)

5. Auto-Pilot Engagement

5.1 Selection

English

If the autopilot is disengaged, the autopilot shall engage (to the PF FGS) when the AP button on the FCP is pressed (providing no higher priority event occurs at the same time).

SMV

SPEC AG(!Is_AP_Engaged -> AX((m_When_AP_Engage_Switch_Pressed.result & m_No_Higher_Event_Than_AP_Engage_Switch_Pressed.result) -> Is_AP_Engaged))

5.2 De-Selection

English

If the autopilot is engaged, the autopilot shall disengage when the AP button on the FCP is pressed (providing no higher priority event occurs at the same time).

SMV

SPEC AG(Is_AP_Engaged -> AX((m_When_AP_Engage_Switch_Pressed.result & m_No_Higher_Event_Than_AP_Engage_Switch_Pressed.result) -> !Is_AP_Engaged))

5.3 Operation

English

If the autopilot is engaged, disengaging the autopilot shall not turn off the FD.

SMV

SPEC AG((Is_AP_Engaged & Onside_FD_On) -> AX(!Is_AP_Engaged -> Onside_FD_On))

6. Mode Selection

6.1 Lateral Modes

6.1.1 Operation

English

The default lateral mode shall be Roll Hold.

SMV

```
SPEC AG((Is_This_Side_Active = 1 & Mode_Annunciations_On &
m_Is_No_Nonbasic_Lateral_Mode_Active.result) -> Is_ROLL_Active)
```

English

Only one lateral mode shall ever be active at any time.

SMV

```
SPEC AG((Is_ROLL_Active -> (!Is_HDG_Active & !Is_NAV_Active & !Is_LGA_Active &
!Is_LAPPR_Active)) & (Is_HDG_Active -> (!Is_ROLL_Active & !Is_NAV_Active &
!Is_LGA_Active & !Is_LAPPR_Active)) & (Is_NAV_Active -> (!Is_ROLL_Active &
!Is_HDG_Active & !Is_LGA_Active & !Is_LAPPR_Active)) & (Is_LAPPR_Active ->
(!Is_ROLL_Active & !Is_HDG_Active & !Is_NAV_Active & !Is_LGA_Active)) &
(Is_LGA_Active -> (!Is_ROLL_Active & !Is_HDG_Active & !Is_NAV_Active &
!Is_LAPPR_Active)))
```

6.1.2 Roll Hold Mode

6.1.2.1 Selection

English

If this side is active and the mode annunciations are on, ROLL mode shall be selected if no other lateral mode is active.

SMV

SPEC AG((Is_This_Side_Active = 1 & Mode_Annunciations_On &
m_Is_No_Nonbasic_Lateral_Mode_Active.result) -> Is_ROLL_Selected)

6.1.2.2 De-Selection

English

If this side is active, ROLL mode shall be cleared when any other lateral mode becomes active.

SMV

SPEC AG(Mode_Annunciations_On -> AX(Is_This_Side_Active = 1 &
m_When_Nonbasic_Lateral_Mode_Activated.result -> !Is_ROLL_Selected))

6.1.2.3 Annunciation

English

The controlled variable "Is_ROLL_Selected" shall be true if and only if ROLL mode is selected.

SMV

SPEC AG(Is_ROLL_Selected <-> ROLL = Selected)

7. Cross Channel Synchronization

7.1 Assumptions

7.1.1 Lateral

English

No more than one lateral mode on the other side will ever be active.

SMV

INVAR (Other_Input_RollSel -> (!Other_Input_HdgSel & !Other_Input_NavAct & !Other_Input_LapprAct & !Other_Input_LgaSel)) & (Other_Input_HdgSel -> (!Other_Input_RollSel & !Other_Input_NavAct & !Other_Input_LapprAct & !Other_Input_LgaSel)) & (Other_Input_NavAct -> (!Other_Input_RollSel & !Other_Input_HdgSel& !Other_Input_LapprAct & !Other_Input_LgaSel)) & (Other_Input_LapprAct -> (!Other_Input_RollSel & !Other_Input_HdgSel& !Other_Input_NavAct & !Other_Input_LgaSel)) & (Other_Input_LgaSel -> (!Other_Input_RollSel & !Other_Input_HdgSel& !Other_Input_NavAct & !Other_Input_LapprAct))

INVAR (Offside_Roll_Selected = 1 -> (!Offside_Hdg_Selected = 1 & !Offside_Nav_Active = 1 & !Offside_Lappr_Active = 1 & !Offside_Lga_Selected = 1)) & (Offside_Hdg_Selected = 1 -> (!Offside_Roll_Selected = 1 & !Offside_Nav_Active = 1 & !Offside_Lappr_Active = 1 & !Offside_Lga_Selected = 1)) & (Offside_Nav_Active = 1 -> (!Offside_Roll_Selected = 1 & !Offside_Hdg_Selected = 1 & !Offside_Lappr_Selected = 1 & !Offside_Lga_Selected = 1)) & (Offside_Lappr_Active = 1 -> (!Offside_Roll_Selected = 1 & !Offside_Hdg_Selected = 1 & !Offside_Nav_Active = 1 & !Offside_Lga_Selected = 1)) & (Offside_Lga_Selected = 1 -> (!Offside_Roll_Selected = 1 & !Offside_Hdg_Selected = 1 & !Offside_Nav_Active = 1 & !Offside_Lappr_Active = 1))

7.2 Operation

7.2.1 General

English

If this side is not active, the mode annunciations shall take its value from the offside FGS.

SMV

SPEC AG(Is_This_Side_Active = 0 -> (Mode_Annunciations_On <-> Offside_Modes_On = 1))

SPEC AG(Is_This_Side_Active = 0 -> (!Mode_Annunciations_On <-> Offside_Modes_On = 0))

7.2.2 Lateral

English

If this side is not active, ROLL mode shall take its value from the offside FGS.

SMV

SPEC AG((Mode_Annunciations_On & Is_This_Side_Active = 0) -> (Is_ROLL_Selected <-> Offside_Roll_Selected = 1))

SPEC AG((Mode_Annunciations_On & Is_This_Side_Active = 0) -> (!Is_ROLL_Selected <-> Offside_Roll_Selected = 0))